VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRE PODPORU METRÍK V PROJEKTOCH VÝVOJA SOFTWARU

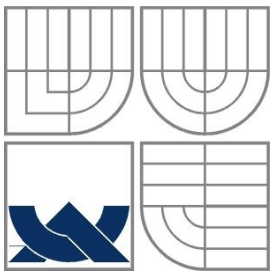
DIPLOMOVÁ PRÁCE

MASTER'S THESIS

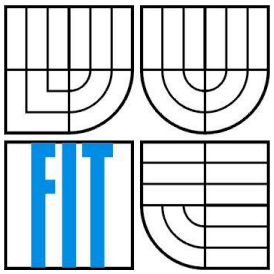
AUTOR PRÁCE *BC. RICHARD REMIÁŠ*

AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SYSTÉM PRO PODPORU METRIK V PROJEKTECH VÝVOJE SOFTWARU

SYSTEM FOR SUPPORING METRICS IN SOFTWARE DEVELOPMENT PROJECTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Richard Remiáš
AUTHOR

VEDOUCÍ PRÁCE — doc. RNDr. Jitka Kreslíková CSc.
SUPERVISOR

BRNO 2012

Abstrakt

Práca sa zaoberá návrhom a implementáciou systému pre podporu metrík, ktorý bude reálne využiteľný pri vývoji softwarového produktu. Je popísaný postup návrhu metód merania a jej aplikácie. V práci sú popísané tri formy vyhodnotenia metrických dát: frekvenčné, časové a vzťahové; spolu s metódami vizualizácie. Nakoniec sú uvedené požiadavky na systém pre podporu metrík a návrh jeho architektúry a samotné detaily implementovaného systému.

Abstract

This work is aimed at design and implementation of system for supporting metrics in software development projects. The procedure of design and application of measurement methods is described. Further metrics data analysis in three domains is described: frequency domain, time domain and relationship domain; together with forms of visualization. Finally, the requirements for system for supporting metrics are enlisted, along with design of architecture and details of implementation.

Klíčová slova

Metrika, vyhodnocení metrik, návrh systému pro podporu metrik, požadavky na systém pro podporu metrik.

Keywords

Metrics, metrics analysis, design of system for supporting metrics, requirements for system for supporting metrics.

Citace

Richard Remiáš: Systém pre podporu metrík v projektoch vývoja softwaru, diplomová práce, Brno, FIT VUT v Brně, 2012.

Systém pre podporu metrík v projektoch vývoja softwaru

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením doc. RNDr. Jitky Kreslíkovej CSc.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Richard Remiáš
18.5.2012

Poděkování

Chcel by som sa poďakovať pani docentke Kreslíkovej za podporu a usmernenie pri vypracovávaní semestrálneho projektu a môjmu priateľovi Ondrejovi Harmanovi za výborné rady ohľadne prezentačného hľadiska.

Obsah

1 Úvod

„Keď to nedokážeš zmerať, nedokážeš to riadiť.“ Voľný preklad výroku Petra F. Duckera, ktorý sa veľkou mierou podieľal na utvorení moderného manažmentu. Táto práca je založená na uvedenej myšlienke. V súčasnosti sa pozornosť výrazne viac upiera na proces výroby softvéru, jeho zefektívnenie, než na prostriedky, ktoré sa k nemu používajú. Efektivita procesu je z veľkej časti daná kvalitou jeho riadenia. Ak chceme zvýšiť produktivitu, musíme vedieť, na ktoré činnosti a aspekty sa zamerať. Práve merania, metriky a ich analýza odhaľujú kritické, problémové, ale aj správne fungujúce kroky výroby softvérového produktu.

Je potrebné v súčasnej dobe zlepšovať riadenie a vývoj softvérových projektov? Existuje ešte priestor, kam sa zlepšovať? Skupina Standish Group každoročne robí prieskum úspešnosti softvérových projektov. V roku 2002 zistili, že iba 26% projektov skončilo včas a s plánovaným rozpočtom, 28% bolo zrušených pred dodaním. Zvyšok prekročil rozpočet, alebo plán a neposkytoval požadovanú funkcionalitu. Z prieskumu v roku 2009 vyplýva, že počet úspešných projektov, skončených v rámci rozpočtu a na čas sa zvýšil na 32% a naopak počet zrušených projektov sa znížil na 24% [1]. Za sedem rokov rozdiel 6% respektíve 4%. Z uvedených údajov vyplýva, že úspešnosť rastie, aj keď len pomalým tempom. Miera úspechu, ktorá sa blíži jednej tretine ale nie je dostatočná. Podľa nej priemerná firma by z troch začatých projektov úspešne v rámci plánu ukončila len jeden. Z ostatných dvoch by jeden buď vôbec nedobehol, alebo oba by skončili v stave, keď firme prinesú iba zlomok plánovaného zisku. Dôsledky sa odrážajú nielen na firme, na jej referenciách a morálke personálu, ale aj na cenách produktov, ktoré musia byť tým pádom vyššie aby bol produkovaný zisk, čo nepoteší ani zákazníkov.

Práca pozostáva zo šiestich kapitol vrátane úvodu a záveru. Druhá kapitola – Získanie údajov – sa venuje metodikám, ktoré určujú postup získavania metrických údajov a návrhu samotných metrík. Tretia kapitola sa venuje analýze získaných údajov, čo je aj hlavná funkcia systému pre podporu metrík. Sú popísané tri prístupy k analýze metrických dát – analýza v časovej, frekvenčnej a vzťahovej doméne. Štvrtá kapitola sa zaoberá špecifikáciou požiadaviek na všeobecný systém pre podporu metrík, návrhom konkrétneho systému a jeho implementáciou. Kapitola päť zhodnocuje dosiahnuté výsledky implementovaného systému pre podporu metrík – MetricsVisage. V závere je zhodnotená celá práca a sú diskutované možnosti budúceho rozšírenia.

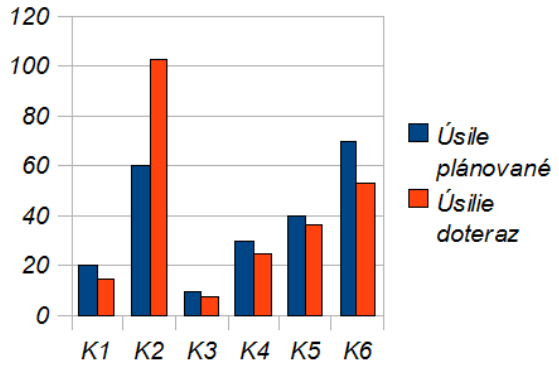
Kapitoly 2, 3 a podkapitoly 4.1 a 4.2 boli vytvorené v rámci semestrálneho projektu, podľa zadania. Diplomová práca sa potom venovala hlavne implementácií systému pre podporu metrík a hodnoteniu výsledkov.

2 Získanie údajov

Ako môžu metriky prispieť k úspešnosti projektov? Za predpokladu, že sa s metrikami pracuje správne môžu nám priniesť veľa úžitku, hlavne v podobe prínosu k riadeniu [2]. Vďaka metrikám získavame lepší pohľad na projekty a aj procesy. Nech máme vo firme bežiaci projekt, na ktorom pracujú tri tímy A, B a C. Každý tím má na starosti viacero častí projektu, ako je ukázané v tabuľke 1.1. Projekt v predposlednej iterácií začne zaostávať za plánom. Vzhľadom na to, že sú k projektu vedené metriky, máme informácie o tom, ktorá jeho časť, respektíve ktorý tým spôsobuje meškanie (graf 2.1).

Tím A	komponenta 1, komponenta 2
Tím B	komponenta 3, komponenta, 4, komponenta 5
Tím C	komponenta 6

Tab. **1.1**: Rozloženie komponent medzi tímy.



Graf **2.1**: Zobrazenie plánovaného a skutočného úsilia na komponentu.

V grafe vidíme, že úsilie vynaložené na zhotovenie komponenty číslo 2 vysoko presahuje plánované úsilie. Z tohto faktu je možné vyvodiť niekoľko záverov: tím A nie je schopný efektívne pracovať na výrobe komponenty č. 2, alebo odhad spravený pre komponentu č. 2 bol nesprávny. V každom prípade má vedenie vďaka metrikám informáciu, ktorú časť procesu treba posilniť a súčasne, ak sa metrické údaje o tomto projekte uložia do firemnej databáze, môže sa v budúcnosti predísť zlému odhadu náročnosti komponenty, ktorá bude podobná komponente č. 2. Ďalej vieme zistiť, ako veľmi sa získané výsledky odchyľujú od záväzkov, ktoré si na začiatku určili jednotlivý manažéri tímov. Medzi ďalšie výhody metrík patria objektívne a opakovateľné rozhodnutia o prideľovaní financií, zlepšenie komunikácie medzi technickým a obchodným/marketingovým manažmentom.

Oplatí sa zavádzať metriky? Nebude to stáť príliš veľa? Každá nová technológia niečo stojí. Aj zavedenie metrík do procesu vývoja produktov niečo stojí. Podľa [2] zavedenie metrického programu do projektu stojí 0,3-1% ceny spojenej s príslušným technologickým úsilím. Čiže ak by sme metriky zavádzali do 200-člennej jednotky, tak na zber a analýzu metrík by bolo potrebné do rozpočtu zahrnúť zhruba 1 človeko/rok, teda jednu osobu, ktorá má metriky na starosti. Pri zavádzaní treba samozrejme rátať s vyššími nákladmi, ktoré sú spojené s školením personálu, vytvorením šablón

a získaním a inštaláciou softvéru pre zber metrík. Cena je výrazne ovplyvnená aj štruktúrou organizácie. Napríklad ak má organizácia firemný portál distribúcia a zber metrík sa podstatne urýchli a nie je potrebné zavádzanie nových prostriedkov.

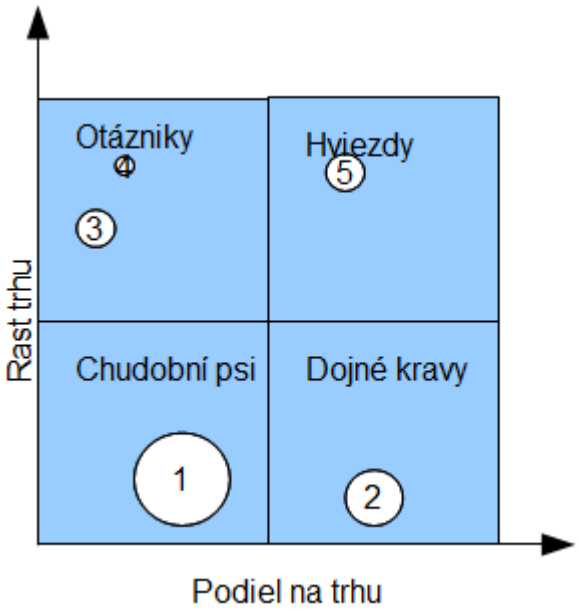
2.1 Postup merania

Postup merania zakladá na troch krokoch: získanie údajov (metrík), vyhodnotenie údajov a vykonanie rozhodnutia založeného na vyhodnotení [2]. Tento cyklus sa počas projektu stále opakuje a musí byť kompletný. Údaje, ktoré získame, ale nevyužijeme, sú strata peňazí. Všetky získané metriky sa musia vyhodnotiť a na ich základe urobiť rozhodnutia. Metriky sa rýchlo „kazia“. Ak ich necháme ležať v dátovom sklade s tým, že sa k nim vrátime neskôr, môže to byť už príliš neskoro. Vtedy už budú údaje neaktuálne vzhľadom na vývoj projektu. Neznamená to ale, že použitých metrík je potom možné sa okamžite zbaviť. Ak by sme ich vymazali, stratili by sme mocný nástroj, časť takzvanej „knowledge base“ organizácie. Na základe údajov nameraných v už ukončených projektoch je možné zvýšiť kvalitu odhadov pre projekty budúce. Čím viac týchto údajov máme k dispozícií, tým lepšia bude presnosť odhadu na nich založeného.

Aby sme mohli údaje použiť, je ich treba najprv získať, preto prvým krokom je meranie. Čo je nutné merať? Je zbytočné merať hodnoty, ktoré nemajú vypovedaciu hodnotu, ktoré nám neprezradia to, čo chceme vedieť. Čo chceme vedieť? Chceme vedieť ako dosiahnuť svoje ciele, preto musíme z cieľov odvodiť, aké informácie budeme potrebovať na overenie ich dosiahnutia a ako tieto informácie získať z činností. Je nutné vytvoriť sadu údajov, ktoré nám budú poskytovať prehľad o projekte a jeho stave. Takéto sady metrík sa v projektoch často zaznamenávajú ako *prístrojová doska* (dashboard). Obsah prístrojovej dosky tvoria informácie typu dodržovanie výkonu oproti plánovaným hodnotám, alebo hodnota projektu v danom okamihu. Tieto údaje nám pomáhajú odhaliť projekty, ktoré nemajú dostatočnú výkonnosť alebo sú vystavené vyššiemu riziku. Prítomnosť týchto informácií upozorní manažérov na prítomnosť problému, ktorý môžu následne začať riešiť. Je dôležité, aby tieto prístrojové dosky mali pre rôzne projekty rovnakú, alebo prinajmenšom podobnú štruktúru. Vymýšľať novú formu pre každý projekt vyžaduje drahocenný čas, ktorý sa môže miesto toho použiť na jeho realizáciu, ale aj zhoršuje vzájomné porovnávanie projektov na vyššej úrovni. Ak všetky projekty dodržujú predpísaný postup, je pomerne jednoduché zistiť ich stav, určiť na ne vplývajúce riziká a riešiť problémy bez toho, aby sa bolo nutné podrobne venovať každému z nich. Metriky ideálne ako indikátory stavu projektu sú miera dodržania rozpočtu/plánu, kvality produktu, alebo prinášaná hodnota. Ak projekt niektorú nespĺňa dostatočne, je to podnet, aby sa ním manažment začal zaoberať a prípadne zvažoval aj jeho zrušenie. Keď sú odhalené všetky dôležité indikátory, je vhodné ich umiestniť na jedno miesto. Zahrnutie často zadávaných databázových otázok do užívateľského rozhrania šetrí čas a snahu. Potom je tvorba správ o projekte, ako aj vyhodnotenie údajov veľmi rýchle a pohodlné.

Už máme potrebné informácie a indikátory identifikované a údaje z projektov sú v dátovom sklade. Teraz prichádza na rad vyhodnotenie týchto údajov. Vyhodnotenie v zmysle hodnoty produktu – pomer ceny a prínosu, vyhodnotenie s ohľadom na trh, v ktorom hotový produkt plánujeme prezentovať, to všetko má vplyv na budúcnosť projektu, očakávania a riziká. Frekvencia vyhodnocovania údajov je kritická. Ako často musíme získané údaje vyhodnotiť aby sme spravili správne rozhodnutia? Perióda závisí na úrovni vyhodnotenia. Na úrovni správy projektov by malo byť týždenné vyhodnotenie predpokladov vhodnou voľbou, čo sa týka vynaloženého úsilia a prínosu. Na vyššej úrovni ako je zhodnotenie portfólia projektov mesačné zhodnotenie prináša dostatočnú pružnosť pri kontrole projektov. Dlhšie intervaly by mohli mať za následok straty. Ak by nastal v rámci projektu problém, po mesiaci by sa už mohol stať neodstrániteľnou súčasťou a následne by mohol viesť k zrušeniu projektu. Takisto z hľadiska portfólia je možné rýchlejšie korigovanie prúdenia zdrojov do projektov – ak nejaký projekt nespĺňa očakávania, alebo cieľový trh sa stane nevhodným, je výhodnejšie ho pozastaviť alebo zrušiť skôr a presunúť úsilie inam. Z tohto dôvodu je

rovnako dôležité vyhodnocovať projekty nielen izolovane, ale aj ako celok, teda súčasť portfólia. Aj keď projekt napreduje podľa plánu, neobsahuje chyby a prináša očakávaný zisk, nemusí sa firme oplatiť ho ďalej živiť. Tu prichádza na rad analýza z hľadiska portfólia. Tu môže pomôcť aj jednoduchá BCG matica (obr. 2.1).



Obr. 2.1: BCG matica projektového portfólia [inšpirované 2].

Najlepšie projekty sa nachádzajú v časti hviezdy, naopak najhoršie, z hľadiska trhu sa nachádzajú v časti chudobní psi. Jednotlivé kruhy predstavujú projekty a ich veľkosť predstavuje hodnotu investícií. Vidíme, že na projekt číslo 1 je vynakladaných veľa investícií a naopak na projekt číslo 5 menej. V tomto stave by bolo ideálne projekt 1 zastaviť a naopak investovať do projektu 5.

Keď máme údaje zanalyzované, prichádza na rad učinenie rozhodnutia o budúcnosti projektu a aj jeho vykonanie. Rozhodnutie môže byť či už na úrovni projektu, alebo na úrovni projektového portfólia, prípadne na iných úrovniach. Dôležité je, aby sa tieto rozhodnutia opierali o získané údaje. Ak by sme ignorovali namerané hodnoty, všetko úsilie vynaložené na ich zber a analýzu by predstavovalo stratenú investíciu a učinené rozhodnutie by bolo s veľkou pravdepodobnosťou nesprávne. Bežať by mali iba tie projekty, ktoré prinášajú najväčšiu hodnotu, a majú najmenší čas návratnosti. Zároveň je vždy dobré mať vytvorenú istú rezervu pre nové projekty, nasadenie nových technológií, riešenie nečakaných udalostí, alebo kríz. Pri riadení projektov existujú tri základné rozhodnutia. Zrušenie projektu je najextrémnejšie, ale netreba ho vynechávať. Ak je už v začiatku projektu viditeľná fatálna chyba, ako nerealistický plán a nízky rozpočet, je úspornejšie projekt zrušiť, než ho nechať bežať a investovať doň viac financií, ktoré sa s veľkou pravdepodobnosťou nenavrátia. Druhá možnosť je zasiahnuť do projektu a zmeniť ho. Zmena sa môže týkať rozpočtu – zníženie alebo navýšenie, plánu, organizácie personálu – pridanie personálu alebo prípadná reorganizácia. Zmeny tohto druhu sú vhodné, keď sa v projekte vyskytnú problémy, ktorých vyriešenie môže priniesť projekt späť na tú správnu koľaj. Posledná možnosť je nechať projekt bežať ako je. Nech sa manažér rozhodne pre ktorúkoľvek z týchto možností, je nutné aby jeho rozhodnutie bolo transparentné a podložené faktami.

2.2 Proces získavania údajov

Prvým krokom v práci s metrikami je, ako bolo uvedené v predošlej kapitole, získanie údajov. Údaje tvoria základ, na ktorom stavajú ostatné kroky, preto musia byť presné a čo najmenej skreslené. Čo je potrebné na získanie údajov? Musíme vedieť čo chceme odmerať. Predmet merania nám určujú stanovené ciele organizácie. Poznať predmet merania avšak nestačí. Je potrebné ho vedieť správne reprezentovať. Je nutné poznať jeho súčasti, mať identifikované všetky atribúty, ktoré majú na meranie vplyv. Okrem predmetu merania musíme mať určenú aj metódu merania, teda spôsob, presne definovaný postup, ako chceme údaje o predmete merania získať. Keď máme definované potrebné komponenty na meranie, zostáva ich implementovať a vykonať samotné získanie údajov.

Proces získania údajov môžeme rozdeliť na dva kroky[3]:

1. Návrh metódy merania
2. Aplikácia metódy merania

Vstupy pre návrh metódy tvorí popis meraného predmetu, súbor pojmov a techník potrebných k samotnému návrhu merania. Výstup tvoria identifikované koncepty a konštrukcie, ktoré budú merané, ako aj princíp ich merania a implementácia metódy merania – spôsob priraďovania numerických hodnôt. Cieľom tejto práce síce nie je návrh metód merania, ale je potrebné poznať postup tvorby takejto metódy, aby bolo možné pri návrhu systému identifikovať vhodné metódy. Výstup prvého kroku – predmet merania a metóda zároveň tvoria vstup kroku druhého. Výstup z druhého kroku, ktorý bude neskôr analyzovaný, tvoria primárne výsledky merania. Okrem výsledkov je vhodné zahrnúť aj indikátory kvality merania – ako presné je vykonané meranie, prípadne faktory, ktoré mohli mať na výsledok merania vplyv.

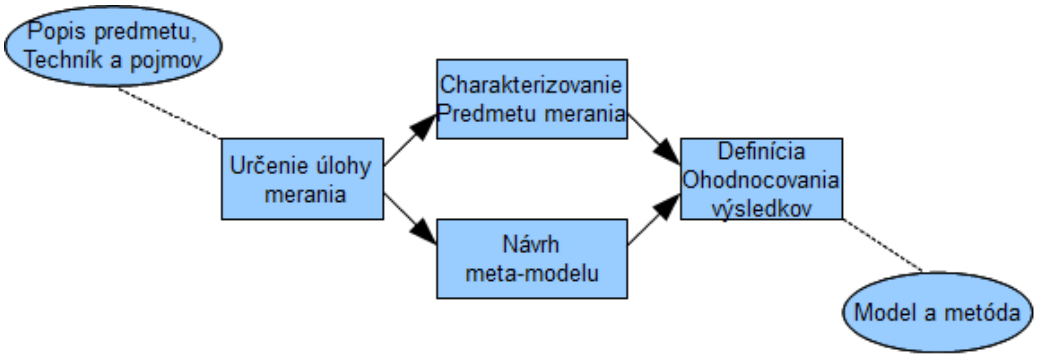
2.3 Návrh metódy merania

Prvý krok získavania údajov – návrh metódy merania sa skladá zo štyroch podkrokov [3]:

1. Určenie úlohy merania.
2. Charakterizovanie predmetu merania, čo zahŕňa špecifikáciu entít, ktoré majú byť merané a ich atribútov, ktoré môžu mať vplyv na meranie.
3. Návrh meta-modelu, na ktorom bude meranie uskutočnené – súbor entít a atribútov a vzťahov medzi nimi.
4. Definícia ohodnocovania výsledkov.

Činnosti 2 a 3 často prebiehajú paralelne (obr. 2.2). Tieto podkroky by mali prebiehať interaktívne, aby bolo možné navrhnutý model postupne zlepšovať a dostávať tak presnejšie výsledky.

V prvom kroku – určenie úlohy merania – odpovedáme na otázky: čo chceme merať? Z akej perspektívy chceme merať? Kto bude výsledky merania využívať? Význam prvej otázky je zjavný – musíme poznať predmet merania, napríklad kvalita programu. Dôležité je aj poznať perspektívu merania, pretože rovnaká vlastnosť môže byť vnímaná rôznymi spôsobmi pre rôzne zainteresované strany. Napríklad kvalitu vníma inak zákazník ako vývojár. Zákazníka zaujíma na koľko splňuje produkt požadovanú funkčnosť alebo s akou pravdepodobnosťou sa môže stať, že systém zlyhá. Naproti tomu vývojár vníma kvalitu z iného hľadiska, ako je napríklad počet chýb, ktoré sa vyskytujú v kóde. Preto treba mať jasne určenú perspektívu merania.



Obr. 2.2: Činnosti návrhu metódy merania [inšpirované 3]

Aby sme boli schopní zmerať predmet merania, ako sme ho v prvom kroku určili musíme ho najprv správne charakterizovať a nadefinovať. Táto charakteristika spočíva v rozložení predmetu merania na súčasti. Každá súčasť je jasne definovaná, a rovnako je definovaná aj jej úloha v predmete merania ako celku. Vzťahy medzi týmito súčasťami popisujeme v treťom kroku návrhu metódy merania. Keď máme určený predmet merania, učíme entitu, na ktorej ju budeme merať (napríklad počet chýb sa meria na entite spustiteľný kód) a atribúty tejto entity, ktoré budeme merať (napríklad počet chýb na tisíc riadkov kódu).

V treťom kroku navrhujeme meta-model. Meta-model predstavuje abstrakciu predmetu merania, ktorá disponuje všetkými potrebnými charakteristikami, prebranými z predmetu merania, aby na nej bolo možné vykonať merania so správnymi výsledkami. Vytvorený meta-model by mal byť generický – je možné ho nasadiť na rôzne programy, nielen na ten, na ktorom bol vyvíjaný a zároveň je nezávislý na kontexte merania. Meta-model musí taktiež zahŕňať aj postup, akým identifikovať atribúty meranej entity, ktoré majú na meranie vplyv. Napríklad metóda COSMIC na určenie funkčnej veľkosti softvéru pracuje s prvkami „Entry“, ktoré reprezentujú pohyb dát v programe. Meta-model pre metódu cosmic teda zahŕňa definíciu typu „Entry“ ako aj spôsob, akým identifikovať všetky entity typu „Entry“. Nakoniec musí meta-model obsahovať definíciu všetkých súčastí a ich role v rámci skúmaného predmetu merania.

Štvrtý krok, definícia ohodnocovania výsledkov, predstavuje spôsob akým budeme mapovať merané vlastnosti predmetu na numerické hodnoty. Vstup pre tento krok je tvorený meta-modelom a charakteristikou predmetu merania. Definícia môže byť popísaná buďto slovne, alebo matematicky. Slovný popis je vhodný pre praktickú aplikáciu metódy a matematický popis slúži na overenie správnosti a konzistencie metódy a aj na určenie operácií nad výsledkami získanými aplikáciou metódy. Súčasne s definovaním ohodnotenia musíme určiť aj jednotky, ktoré v metóde merania použijeme.

V rámci krokov a podkrokov návrhu metódy merania prebiehajú dve významné aktivity: **definícia princípu merania** a **určenie metódy merania**.

2.4 Definícia princípu merania

V princípe merania by malo byť zahrnuté naše chápanie predmetu merania a všetky znalosti o predmete merania. Inak povedané, výstup tejto aktivity tvorí odpoveď na otázku: čo chceme merať? V súvislosti s obr. 2.2 môžeme povedať, že daná aktivita prebieha hlavne v krokoch *charakterizovanie predmetu merania* a *návrh meta-modelu*.

Vo vyspelých disciplínach je princíp merania definovaný na základe experimentov a zakladá si na existujúcich, dobre overených zákonoch a teóriách. Metódy merania založené na teóriách a zákonoch sú použiteľné nielen v praxi, ale aj na spätné overenie použitých podkladov. V súvislosti so softvérom hovoríme iba zriedka o zavedených, overených teóriách a zákonoch. Z hľadiska teoretických základov je softvérová doména menej vyspelá ako tradičné vedy, preto návrh metód merania prináša so sebou ďalšie problémy. Nielen že musíme vytvoriť použiteľnú metódu, ale často aj vytvoriť a overiť teóriu, na ktorej zakladáme, čo spotrebuje nemalé úsilie a je mimo rámca návrhu metódy merania.

Pri tvorbe metód merania pre softvér sa často vychádza z **reprezentatívnej teórie merania** [4]. Podľa tejto teórie je meranie definované ako mapovanie z *empirického* sveta (relačný systém) do *numerického* sveta pri zachovaní relácií. Reprezentatívna teória merania nám pomáha formalizovať našu intuíciu o tom, ako svet funguje. Namerané hodnoty by mali charakterizovať atribúty entít, ktoré sme merali, pričom zachovávajú relácie, ktoré pozorujeme medzi týmito entitami.

Spomínaná teória má však nedostatky. Nepopisuje presne spôsob ako charakterizovať jednotlivé svety a mapovanie medzi nimi, zakladá hlavne na intuícií, preto je nutné doplniť tieto definície. Do definície princípu merania spadá popis empirického a numerického sveta. Definícia metódy mapovania spadá pod určenie metódy merania.

Popis empirického sveta. Vzhľadom na to, že výsledky merania sú mapované atribúty, je nutné v prvom rade charakterizovať meraný atribút. Atribút musí byť popísaný dostatočne presne a jasne. Charakteristika atribútu vedie k vytvoreniu modelu atribútu. Model musí vymedzovať, čo do atribútu ešte patrí, a čo už je za hranicou skúmania. Je veľmi dôležité, aby tieto hranice boli zvolené tak, aby bol dosiahnutý konsenzus. V softvérovej doméne sa často stretávame so situáciou, keď sú atribúty definované odlišnými autormi odlišne, niekedy aj nepresne. Tejto situácii je nutné predísť práve dosiahnutím konsenzu, ktorý by časom mohol viesť aj k zavedeniu štandardov.

Aby sme mohli presne charakterizovať atribúty, je nutné najprv určiť entitu, ktorej atribút skúmame. V softvérovej doméne sa môžu nejasnosti vyskytnúť aj tu. Napríklad zložitosť sa môže týkať algoritmu, ale aj kódu, ktorý ho implementuje, v zmysle náročnosti operácií. Preto by modely atribútov mali byť generalizované, aby podobné nejasnosti nepredstavovali problém.

Použité modely musia spĺňať dve často protichodné požiadavky:

1. model by mal mať teoretický základ, ktorý je precízny a kompletný,
2. model by mal mať praktický, jednoduchý na použitie a umožniť jednoduchú implementáciu metód merania.

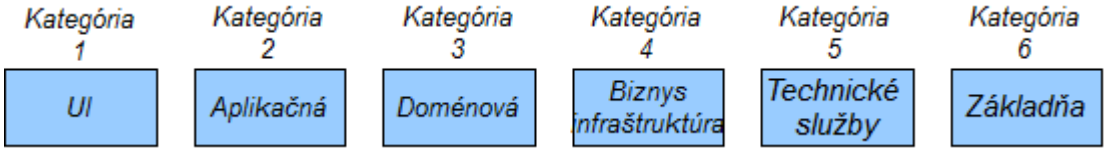
Prvú podmienku najviac spĺňajú matematické modely. Svet by sme mohli modelovať ako matematickú štruktúru $M = (Q, R, O)$, kde Q reprezentuje množinu entít, R množinu relácií a O množinu operácií nad entitami z Q. Pri matematických modeloch máme k dispozícií aj matematické axiómy. Tieto axiómy môžu popisovať niektoré vlastnosti entít alebo operácií. Axiómy taktiež môžu slúžiť ako solídny základ pre formálne dokazovanie správnosti metódy merania založenej na matematickom modeli. Matematické modely sú vhodné aj z hľadiska mapovania. Ak máme *empirický svet* modelovaný matematickým modelom, je jednoduché ho mapovať na *numerický svet*. Nevýhoda matematického modelu spočíva v neschopnosti dostatočne popísať a identifikovať entity *empirického sveta*. Taktiež môže predstavovať problém jeho pochopenie alebo zložitosť implementácie.

Naproti tomu konceptuálne modely sú prijateľné hlavne z pohľadu užívateľskej prívetivosti. Konceptuálne modely sa uplatňujú hlavne v softvérovom inžinierstve (Unified Modelling Language). Princíp konceptuálneho modelovania spočíva v identifikácií základných konceptov a vzťahov medzi nimi a na ich základe sa vybuduje zrozumiteľný model. Často sa používa grafická reprezentácia modelov. Grafická reprezentácia umožňuje spojiť informácie o entitách a atribútoch, ktorými

disponujeme, so štrukturálnou reprezentáciou vzťahov medzi entitami, čo pomáha porozumeniu modelu. Vzhľadom na to, že sa orientujeme na atribúty, ktoré majú byť merané, predstavuje konceptuálny model formu abstrakcie entít – v modeli sú zahrnuté len informácie, ktoré nás zaujímajú.

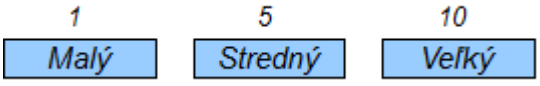
Popis numerického sveta. Popis numerického sveta sa týka hlavne problematiky použitej stupnice. Použitá stupnica nám udáva relácie, aké sa môžu vyskytovať medzi prvkami a operácie nad prvkami. Prvý problém pri výbere stupnice, ktorý musíme riešiť, je existencia vhodnej stupnice. Tento problém sa nazýva reprezentačný problém a bol už vyriešený pre mnoho relačných systémov. Druhá situácia, ktorá môže nastať je, že existuje viacero vhodných reprezentácií (stupníc). Potom je treba riešiť problém ako vybrať tú najvhodnejšiu. *Empirický svet* je podľa reprezentatívnej teórie merania relačným systémom, takisto ako je aj *matematický svet*. Relačný systém môžeme ohodnotiť na základe „bohatosti“. Jeden relačný systém je „bohatší“ ako druhý, ak obsahuje všetky relácie druhého relačného systému. Najčastejšie používané stupnice (matematický svet) sú: nominálna, ordinálna, intervalová, pomerná, absolútna. Stupnice sú usporiadané podľa „bohatosti“ od najmenej „bohatej“ po najviac „bohatú“. Na základe „bohatosti“ vieme určiť aj stupnicu vhodnú pre model *empirického sveta*. Čím je model „bohatší“ tým „bohatšiu“ stupnicu vyberieme pre jeho reprezentáciu.

Nominálna stupnica. Nech máme určité triedy. Každú meranú entitu zaradíme podľa hodnoty skúmaného atribútu do jednej z týchto tried. Na uvedenom princípe je založená nominálna stupnica. Prvky v nominálnej stupnici nie sú usporiadané, iba kategorizované do jednotlivých tried. Nominálna stupnica nepodporuje žiadne operácie, jedine porovnanie na náležitosť do určitej triedy. Príklad nominálnej stupnice je začlenenie balíčkov a tried do vrstiev architektúry vo fáze návrhu informačného systému (obr. 2.3).



Obr. 2.3: Kategórie rozčlenenia balíčkov a tried v architektúre IS.

Ordinálna stupnica. V ordinálnej stupnici je k rozčleneniu do skupín z nominálnej stupnice pridané usporiadanie. Jednotlivé triedy sú usporiadané podľa meraného atribútu. Použité mapovanie musí zachovávať usporiadanie. Táto stupnica stále neposkytuje ďalšie operácie ako napríklad sčítanie alebo odčítanie. Príkladom takejto stupnice môžu byť kategórie chýb v projekte s ohľadom na dopad na cenu projektu (obr. 2.4). Pri tvorbe mapovacej funkcie si musíme dávať pozor na ohodnotenie kategórií. V našom príklade podľa daného usporiadania nemôže nastať, aby kategória „stredný“ mala vyššie ohodnotenie ako kategória „veľký“.



Obr. 2.4: Kategórie závažnosti chýb s ohľadom na dopad na cenu s ohodnotením.

Intervalová stupnica. Prínos intervalovej stupnice oproti dvom predošlým spočíva v pridaní hodnoty intervalu, o ktorý sa dve kategórie líšia. Tým pádom vieme zachytiť veľkosť skoku, o ktorý sa dve kategórie od seba líšia. V tejto stupnici existujú operácie sčítania a odčítania, avšak nie je možné deliť ani násobiť, pretože to nemá zmysel. Intervalovú stupnicu môžeme aplikovať pri menších zmenách na predošlý príklad (obr. 2.5). Všetky kategórie chýb sa líšia o rovnakú hodnotu. Môžeme povedať, že ak závažnosť chyby na základe novo - odhalených faktorov stúpne o 1 stupeň, posunie sa do vyššej

kategórie. Intervalové stupnice podporujú afinitné transformácie, pre ktoré platí, že vždy môžeme nájsť také čísla a a b, že platí $S = aS' + b$, pričom S aj S' sú intervalové stupnice.

Obr. 2.5: Kategórie závažnosti chýb s ohľadom na dopad na cenu s intervalovým ohodnotením.

Pomerná stupnica. Pomerná stupnica nám prináša okrem aditívnych operácií aj operácie s pomerom, aby sme mohli povedať, že jedna entita má dvakrát viac atribútu než druhá entita. Všeobecne si pomerová stupnica zachováva vlastnosti predošlých, teda usporiadanie a jednotnú veľkosť intervalu medzi entitami, plus možnosť tvorby pomerov, existenciu nulového prvku a podporu aritmetických operácií. Túto stupnicu môžeme použiť napríklad pre meranie veľkosti kódu v LoC (Lines of Code – riadky kódu). Nulový prvok predstavuje prázdny kód, jednotka prírastku je jeden riadok a ak má jeden súbor 10 riadkov a druhý 20, môžeme povedať že druhý je dvakrát taký veľký ako prvý.

Absolútna stupnica. Absolútna stupnica podporuje rovnaké operácie ako pomerná stupnica, ale je unikátna. To znamená, že existuje iba jedna možnosť ako môžeme uskutočniť meranie. Meranie môžeme uskutočniť iba počítaním elementov s danou vlastnosťou. Ako príklad je možné uviesť počítanie chýb pri testovaní časti projektu. Počet chýb môže byť meraný len jedným spôsobom a to spočítaním všetkých vyskytnutých chýb.

2.5 Určenie metódy merania

V predošlej podkapitole sme popísali empirický a matematický svet. Na základe týchto popisov nasleduje určenie mapovania medzi týmito dvomi svetmi. Mapovanie zodpovedá samotnej metóde merania.

Z matematického hľadiska, čím máme o mapovaní menej informácií, tým viac sa bude štruktúra matematického sveta približovať štruktúre empirického sveta. Naopak, ak máme príliš veľa informácií o mapovaní môže nastať situácia, že máme jednotlivé dvojice z empirického a matematického sveta spolu s ohodnoteniami, ktoré reprezentujú výsledok merania. Takýto prípad nie je priaznivý pre praktické použitie – viac sa podobá tabuľke, ako algoritmu. Na druhú stranu je nepravdepodobné, že by tento prípad nastal, lebo v praxi často meriame potenciálne nekonečné množiny entít.

Z hľadiska nasadenia metóda merania predstavuje postupnosť operácií, ktoré musia byť vykonané v procese merania. Táto postupnosť operácií musí korešpondovať s definovaným mapovaním a to tak, že hovorí ako mapovať danú entitu na jej príslušnú hodnotu, pri dodržaní hodnotiacich pravidiel (podkapitola 2.1). Teda hodnotiace pravidlá nám hovoria, ako nájsť entity, ktoré nás zaujímajú, a ktoré z ich atribútov brať do úvahy. Následne podľa definície mapovania určíme postupnosť operácií, ktorými daný atribút ohodnotíme príslušnou hodnotou.

V prípade, že meranie zahŕňa počítanie entít, tak musí obsahovať pravidlá na určenie entít, ktorých sa meranie týka, pravidlá, podľa ktorých je možné vylúčiť ostatné entity, postup počítania a podobne. Ak sa výsledok merania získava z iných hodnôt dopočítaním, tak nám určuje spôsob dopočítania požadovaných hodnôt.

Ako príklad postupu je možné uviesť metódu merania energetickej hodnoty. Merať energetickú hodnotu objektu nie je možné priamo, ale je možné ju zistiť jej využitím. Metóda môže byť založená napríklad na prenose energie horením. Objekt vložíme do dokonale izolovanej nádoby a zapálime.

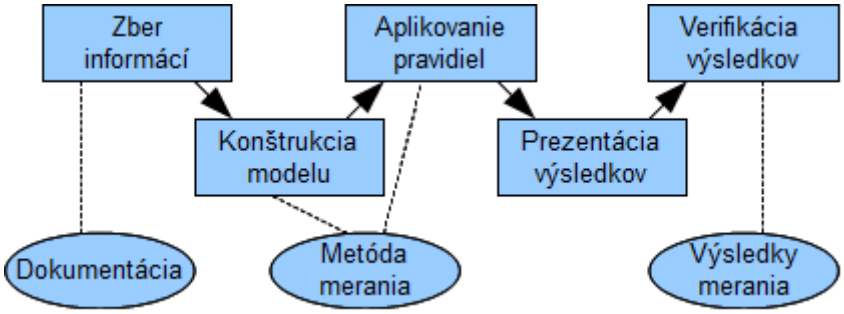
Nad touto nádobou je druhá izolovaná nádoba s vodou a teplomerom. Potom energiu objektu môžeme dopočítať na základe rozdielov teploty vody pred zapálením objektu a po zapálení objektu. Toto meranie musí byť samozrejme podložené všetkými príslušnými fyzikálnymi zákonmi a musí byť uvedená aj existencia chyby, ktorá vyplýva z reálnych vlastností materiálov.

2.6 Aplikácia metódy merania

Ako náhle máme metódu merania kompletne navrhnutú, alebo máme k dispozícii už existujúcu metódu a všetky potrebné prostriedky, je možné aplikovať ju na konkrétny program. V terminológií merania znamená aplikácia metódy v konkrétnom kontexte *návrh meracej procedúry*. Meracia procedúra je oproti meracej metóde viac špecifická – konkrétnejšie sa zaoberá praktickou stránkou mapovania v súvislosti s konkrétnym programom. Dokument, ktorý ju reprezentuje je *správa o meraní*, popisujúca konkrétnu implementáciu metódy. Ak meranie zahrňuje nejaký nástroj (v prípade softvéru program), správa zahŕňa špecifikáciu použitého nástroja, možnú odchýlku a podobne. Na tejto úrovni je dôležitý práve kontext merania. Prinajmenšom musí zahŕňať úlohu merania a podmienky, za ktorých je meranie vykonané. Podmienky sa týkajú priamo aplikácie merania, alebo aj úrovne znalostí, ktoré máme o meranom atribúte, prípadne kvalitu použitých modelov. Posledné dve spomínané často nie sú v softvérovej doméne na vysokej úrovni vyspelosti, resp. často nebývajú jednotné (zložitosť programu, kvalita, ….).

Aplikácia metódy merania je vykonávaná v štyroch krokoch, ktoré sú zobrazené aj na obr. 2.6 :

1. zber relevantných informácií o software
2. konštrukcia modelu, ktorý má byť meraný
3. aplikácia hodnotiacich pravidiel
4. prezentácia výsledkov
5. verifikácia výsledkov



Obr. 2.6: Graf postupnosti krokov aplikácie metódy merania [inšpirované 3]

Zber informácií o meranom software je vykonávaný na software samotnom, alebo ak software ešte nie je dostupný, na softwarovej dokumentácií. Informácie získané v tomto kroku pomáhajú pri vytváraní modelu v kroku druhom.

Konštrukcia modelu nastáva po ukončení zberu potrebných informácií. Zozbierané informácie sú aplikované na definície modelov a sú vytvorené modely špecifické pre daný software, na ktorých bude meranie prebiehať. Použité modely pochádzajú z návrhu metódy merania (podkapitola 2.1). Ak je model už vyhotovený v dokumentácií, tento krok sa preskakuje.

Aplikovanie hodnotiacich pravidiel predstavuje priradenie numerických hodnôt relevantným atribútom v meraných entitách.

Prezentácia výsledkov zobrazuje výsledky merania získané v predošlom kroku v prehľadnej forme. Dokumentáciu zvyčajne okrem samotných výsledkov tvoria jednotky merania, popis postupnosti krokov samotného merania, popis procedúry merania a podobne.

Verifikácia výsledkov je posledný krok merania. Nie je dostačujúce výsledky získať, ale aj overiť správnosť výsledkov. Tradičné metódy sú napríklad overenie matematických výpočtov, porovnanie s výsledkami podobných meraní na minulých projektoch a podobne.

3 Vyhodnotenie údajov

3.1 Frekvenčné vyhodnotenie údajov

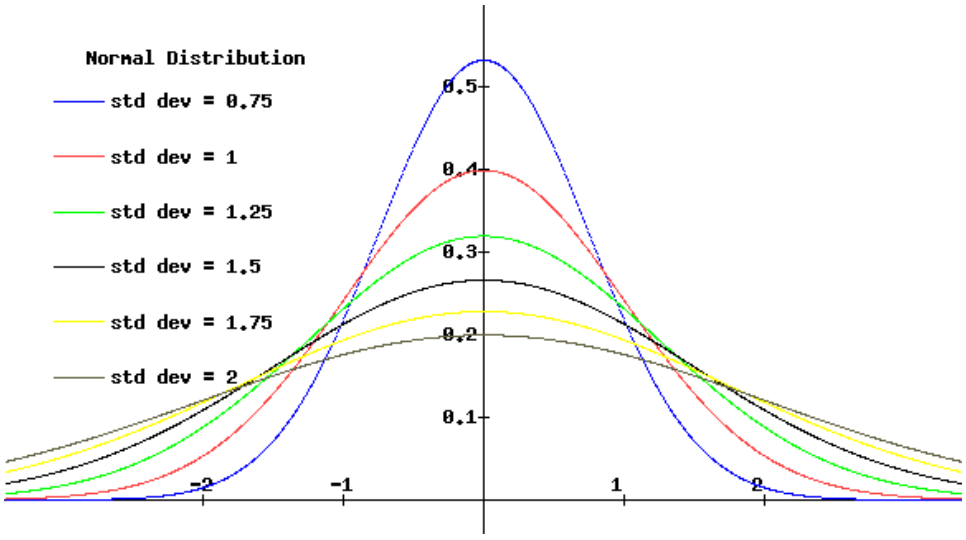
Vyhodnotenie metrických údajov vo frekvenčnej oblasti pomáha pri odhaľovaní odchýlok v procesoch, ako aj pri porozumení samotných procesov. V organizácii s procesným riadením sú rovnaké procesy vykonávané opakovane. Ak nazhromaždené údaje z vykonaných behov procesov analyzujeme (frekvenčná analýza), dostaneme unikátnu charakteristiku procesu. V takejto charakteristike sú jasne viditeľné hlavné črty procesu, ako aj odchýlky vyskytujúce sa v procesoch.

Výstupom vyhodnotenia metrických údajov vo frekvenčnej oblasti je *distribučná funkcia*. Práve táto funkcia reprezentuje unikátnu charakteristiku procesu, obsahujúcu jeho hlavné črty. Vyhodnotenie distribučnej funkcie je zväčša založené na porovnaní so zavedenými rozloženiami. Najpoužívanejšie rozloženia v informačnej doméne [5] sú *normálne rozloženie* a *rayleighovo rozloženie*.

Normálne rozloženie sa vyskytuje v bežnom živote najčastejšie. Vzorec distribučnej funkcie normálneho rozloženia pre veličinu *x* je:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad (3.1)$$

Kde μ je stredná hodnota a σ je smerodajná odchýlka (sa označuje ako rozptyl). Charakteristickou črtou normálneho rozloženia je, že väčšina výskytov je koncentrovaná okolo strednej hodnoty, a zároveň počet výskytov s hodnotou menšou než stredná hodnota je rovnaký než počet výskytov s hodnotou väčšou než je stredná hodnota (obr. 3.1). Tvar krivky je daný smerodajnou odchýlkou. Čím je väčšia, tým je krivka „sploštenejšia“, naopak so zmenšujúcou sa hodnotou odchýlky sa krivka „špicatí“.

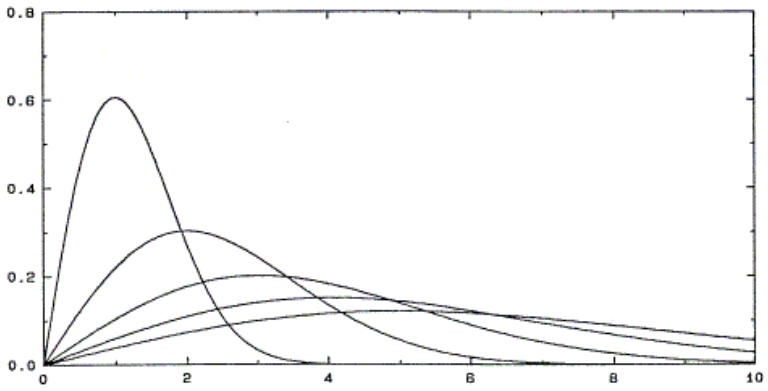


Obr. 3.1: Normálne rozloženie a vplyv zmeny odchýlky [6].

Rayleighovo rozloženie sa líši od normálneho rozloženia tým, že nemá symetrický tvar (obr. 3.2). Vzorec distribučnej funkcie rayleighovho rozloženia je:

$$f(x) = \frac{x}{s^2} e^{-\frac{x^2}{2s^2}} \qquad (3.2)$$

Kde *s* určuje tvar krivky (vypočítaný zo smerodajnej odchýlky). Fakt, že rozloženie je naklonené na jednu stranu, je dôvodom, prečo je tento druh rozloženia používaný. Systémy s ľudským faktorom často vykazujú známky nesymetrickosti. Táto odchýlka je spôsobená tým, že pri riešení problému je možný výber z viacerých možností a práve využitie inej ako ideálnej možnosti vedie k podobnej deformácii.



Obr. 3.2: Rayleighovo rozloženie s rôznym parametrom *s* [7].

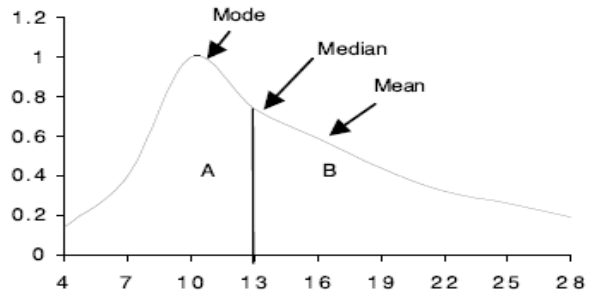
Vyhodnotenie získaných údajov z meraní je vhodné začať *deskriptívnymi štatistikami*. Deskriptívne štatistiky obsahujú nasledujúce parametre [5]:

- priemer
- štandardná chyba (priemeru)
- medián
- modus
- smerodajná odchýlka
- rozptyl
- strmosť
- šikmosť
- rozsah
- suma
- počet hodnôt
- najväčšia hodnota
- najmenšia hodnota

Priemer chápeme ako aritmetický priemer nameraných hodnôt. Pri normálnom rozdelení sa zhoduje s mediánom, v niektorých prípadoch aj s modusom – stredná hodnota. Pri rozloženiach, ktoré nie sú symetrické sa tieto tri hodnoty nezhodujú (obr. 3.3) a nemusí byť vždy ideálne použiť priemer ako indikátor – môže dôjsť k skresleniu údajov. V takomto prípade je vhodnejšie použiť **medián** alebo **modus**. Medián rozdeľuje usporiadanú postupnosť hodnôt na dve rovnaké polovice. Modus reprezentuje hodnotu, ktorá sa v množine nameraných hodnôt vyskytuje najčastejšie. Medián je vhodné použiť najmä vtedy, ak sú v súbore obsiahnuté extrémne hodnoty, ktoré skresľujú priemer. Priemer, medián a modus spolu popisujú *centrálnu tendenciu* procesu.

Smerodajná odchýlka, **rozptyl** a **rozsah** spolu určujú *mieru rozptýlenia*. Smerodajná odchýlka je druhou odmocninou rozptylu. Ako je možné vidieť na obrázkoch 3.1 a 3.2 smerodajná odchýlka veľkou mierou určuje tvar krivky. Čím má väčšiu hodnotu, tým je krivka plochejšia a naopak. Plochejšia krivka znamená viacero odchýlení od strednej hodnoty. V kontexte procesov veľké odchýlky znamenajú malú vyspelosť procesu. Procesy s veľkými odchýlkami sú menej predvídateľné. Tendencia pri zlepšovaní vyspelosti procesov je eliminácia vychýlení od strednej hodnoty procesu. Rozsah predstavuje rozdiel medzi najväčšou a najmenšou nameranou hodnotou. Je potrebné uvažovať rozsah v kontexte odchýlky a rozptylu, pretože okrajové hodnoty bývajú často extrémy so zriedkavým výskytom.

Šikmosť určuje mieru asymetrie. Čím väčšia je šikmosť tým menej symetrická je distribučná funkcia. V praxi to pre proces znamená, že existuje väčší počet odchýlok podobného charakteru. Zároveň nám znamienko šikmosti určuje vzťah mediánu a priemeru. Pre kladné hodnoty je priemer väčší než medián a naopak. **Strmosť** popisuje aké výrazné sú vrcholy krivky.



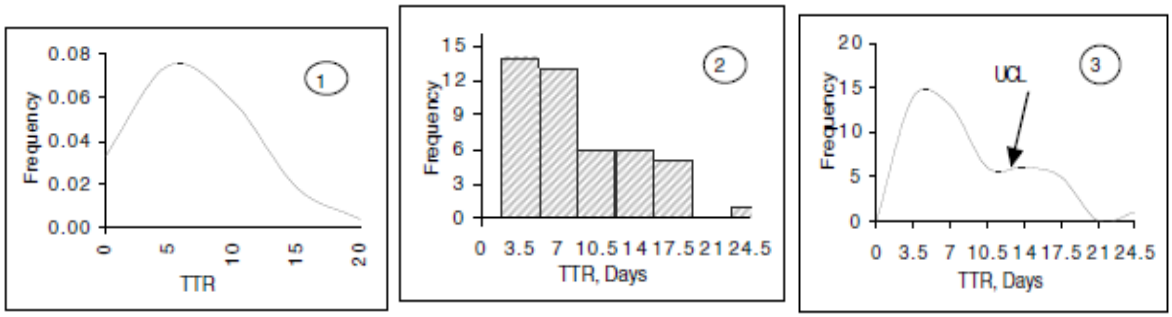
Obr. 3.3: Medián (median), priemer (mean) a modus (mode) na asymetrickej krivke [5].

Ako bolo napísané v úvode podkapitoly, jedným z výstupov, ako aj prostriedkov frekvenčného vyhodnotenia je distribučná funkcia, ktorá funguje aj ako vizualizácia. Okrem distribučnej funkcie je možné frekvenčné dáta vizualizovať aj pomocou *funkcie hustoty pravdepodobnosti* a *histogramu*. Z praktického hľadiska majú najväčšiu výpovednú hodnotu histogram a distribučná funkcia.

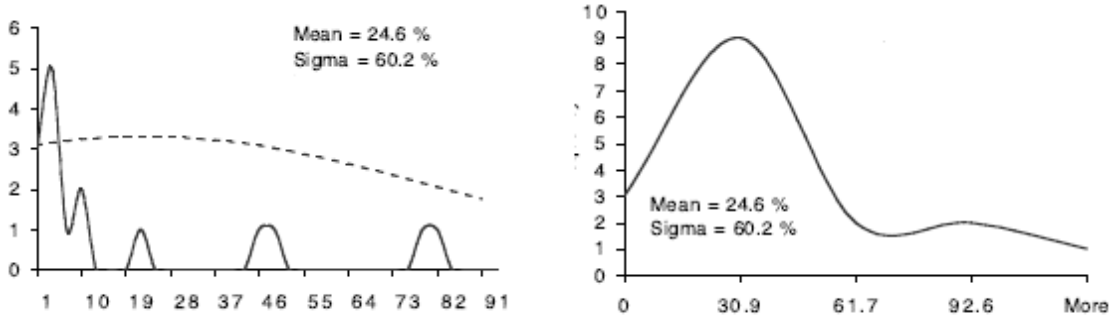
Histogram je pomerne jednoduché zostrojiť. Histogram znázorňuje početnosť prvkov s rovnakou hodnotou. Ak prvky rozdelíme podľa hodnoty do intervalov, môžeme získať histogramy s rôznou hrubosťou. Takto môžeme manipulovať zobrazenie k zvýrazneniu detailov, alebo celkového charakteru. Hodnoty, resp. intervaly sú značené na x-ovej osi a frekvencia na y-ovej osi.

Distribučnú funkciu je možné vyvodiť alebo z histogramu alebo aplikáciou strednej hodnoty a smerodajnej odchýlky na funkciu hustoty pravdepodobnosti. V druhom prípade dostaneme *normálnu distribučnú krivku* podľa zvoleného rozloženia, ktorá nezodpovedá presne reálnemu stavu. Ak chceme vytvoriť distribučnú funkciu z histogramu, preložíme krivku vrcholmi histogramu. Takto získame *empirickú distribučnú funkciu*. Na obrázku 3.4 sú znázornené obe distribučné krivky a histogram pre jeden súbor dát.

Keď máme stanovené predmety skúmania, je nutné uviesť metódy skúmania. Prvou metódou je **frekvenčný sken** [5]. Frekvenčný sken spočíva v menení veľkosti intervalov histogramu. Podľa toho dostaneme odlišné empirické distribučné krivky, ktoré zvýrazňujú odlišné charakteristiky. Na obrázku 3.5 sú znázornené dve krivky toho istého súboru dát s rôznym počtom intervalov. V ľavom obrázku je krivka založená na histograme s 32 intervalmi a vpravo krivka založená na histograme so 7 intervalmi. Z obrázku vľavo môžeme vyčítať, že v procese existujú hlavne výskyty s nízkymi hodnotami. Okrem toho existujú isté skupiny, ktoré majú vyššie hodnoty, teda proces sa odchyľuje od normálneho chovania. Čiarkovane je vyznačená normálna distribučná krivka, ktorá je v tomto prípade výrazne sploštená, čo značí nestabilný proces. Obrázok vpravo nám poskytuje iný pohľad. Nezvýrazňuje konkrétne odchýlky, prípadné problémy, ale celkový obraz o odchýlkach v procese.



Obr. 3.4: Distribučná funkcia, histogram a empirická distribučná funkcia [5].



Obr. 3.5: Distribučné funkcie založené na histogramoch s odlišným počtom intervalov [5]

Analýza histogramov [5] je ďalšou možnosťou frekvenčného vyhodnotenia. Histogram nesie veľa informácií o procese. Pomocou neho je možné určiť napríklad stabilitu procesu, alebo jeho sklon. Príklady možných histogramov sú:

- histogram tvaru U
- histogram s dominantným vrcholom v strede
- hrebeňový histogram
- histogram naklonený doprava/doľava
- bimodálny histogram s porovnateľnými vrcholmi
- bimodálny histogram s jedným dominantným vrcholom
- histogram s viacerými zhlukmi
- plochý histogram
- histogram s dlhým chvostom
- a ďalšie.

Každý z týchto histogramov určuje isté vlastnosti procesu, z ktorého pochádzajú hodnoty použité na vytvorenie histogramu. Procesy s hrebeňovým alebo plochým histogramom bývajú plné neurčitostí. Pri takýchto procesoch je ťažké, až nemožné určovať ich chovanie – známka nevyspelosti.

Bimodálny histogram značí, že v procese sú dve významné hodnoty. Pri procesoch s týmto typom histogramu je nutné dávať si pozor na použitie priemeru. Vzhľadom na to, že histogram obsahuje dva významné vrcholy, bude priemerná hodnota pravdepodobne niekde medzi nimi. Tým pádom môže dôjsť k skresleniu údajov. Napríklad ak vyhodnocujeme metriku odchýlka od plánovaného množstva úsilia. Ak existujú dva vrcholy, každý na opačnej strane od hodnoty nulovej odchýlky, ktoré majú

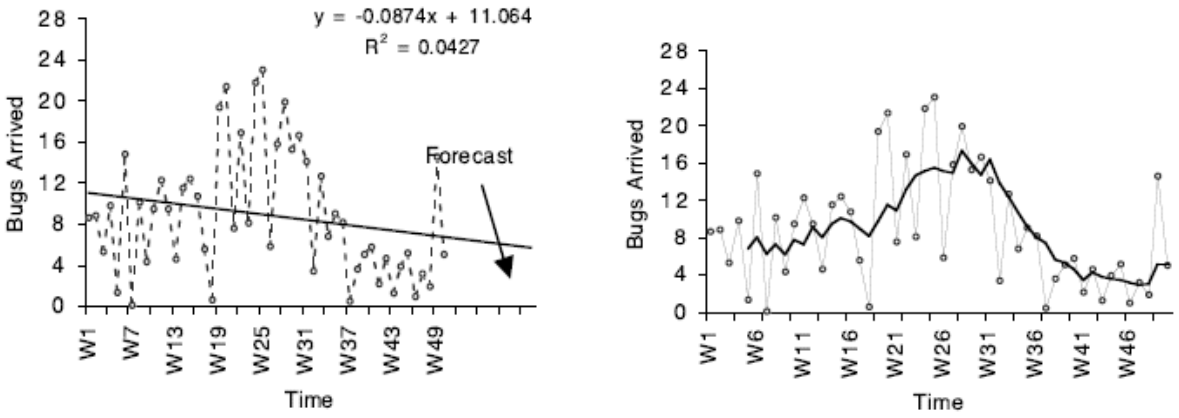
porovnateľnú veľkosť, ich priemer bude vychádzať nulový. Avšak v procese predpokladania potrebného úsilia dochádza k vážnemu problému – problémy bývajú buďto nadhodnotené, alebo podhodnotené.

Histogram s viacerými zhlukmi môže značiť existenciu určitých tried. Napríklad ak meriame produktivitu zamestnancov v metrike LoC (Line of Code) za jednotku času. Každý zhluk predstavuje istú výkonnostnú triedu, do ktorej patrí daný zamestnanec.

3.2 Časové vyhodnotenie

Časové vyhodnotenie [5] – usporiadanie metrických dát podľa časovej následnosti – prináša reálny pohľad na stav sledovanej entity, či už je to projekt alebo proces. Chronologické usporiadanie je prirodzené, pretože všetky procesy prebiehajú v čase. Okrem reálneho pohľadu na stav nám chronologické usporiadanie prináša aj možnosť porovnania súčasného stavu so stavmi z minulosti, alebo aj zhodnotiť vývoj projektu, resp. procesu ako celok. Okrem spätnej analýzy dát môžeme na základe odpozorovaných trendov a charakteristík predpovedať aj pravdepodobný budúci vývoj. Vzhľadom na to, že softwarové projekty sú väčšinou vypracovávané na základe modelov životného cyklu, skladajú sa z určitých etáp, ktoré majú pevne daný začiatok a koniec. Vyhodnotenie metrických údajov v časovej doméne oproti plánom jednotlivých etáp slúži ako vhodný indikátor stavu projektu. Vďaka porovnaniu je možné v projektoch určiť kritické body, podporiť proces rozhodovania a získať reálny pohľad na aktuálny stav projektu.

Pre predvídanie budúceho stavu zo zaznamenaných dát je v prvom rade nutné vyčítať z nameraných dát časové vzory.



Obr. 3.6: Ukážka časových vzorov pre odhalené chyby v programe [5]

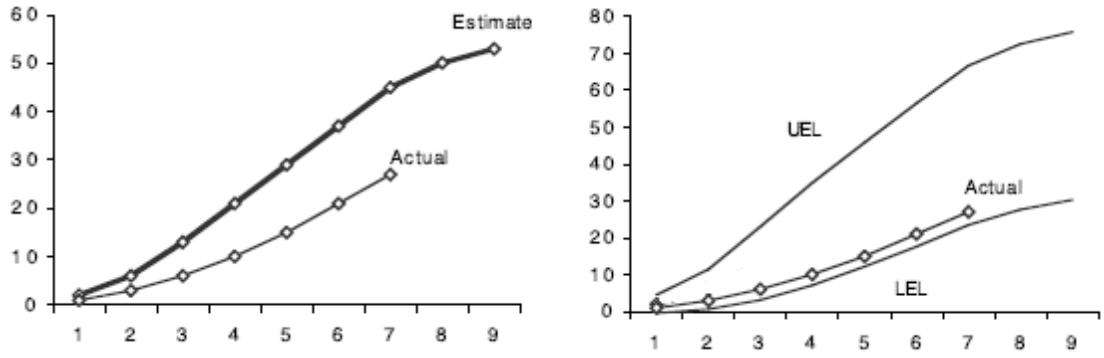
Na obrázku 3.6 je znázornený záznam o odhalených chybách v programe počas jeho vývoja. Na obrázku vľavo je popri zaznamenanom počte chýb zaznamenaný lineárny trend vývoja chýb. Na obrázku vpravo je znázornený posúvajúci sa priemer, počítaný zo štyroch nameraných hodnôt. Z grafu je možné vyčítať, že počet odhalených chýb má klesajúci charakter. Vzhľadom na postupné dokončievanie programu je tento charakter prirodzený – čím viac sa projekt blíži k zhotoveniu, tým menej chýb by sa v ňom malo vyskytovať. Odhalený vzor svedčí o správnom smere projektu. Zároveň na základe klesajúceho počtu chýb je možné zmeniť množstvo prostriedkov, ktoré sú rezervované na ich opravu. Je možné zmenšiť počet ľudí nasadených na opravu chýb, prípadne zmenšiť plánované úsilie a získanú silu zamerať na inú časť projektu.

Pre charakterizovanie správania sa procesu je vhodné použiť pokročilejšie formy grafov zaznamenávajúce časovú následnosť ako napríklad grafy centrálnej tendencie, grafy odchýlky/rozsahu, alebo kontrolné grafy.

Grafy centrálnej tendencie zaznamenávajú priebeh zmien hodnôt ako sú medián, alebo priemer. Je vhodné použiť grafy tohto typu, ak jednotlivé vzorky z merania v sebe obsahujú viacero hodnôt. Príkladom takejto situácie je metrika počet odhalených chýb za týždeň. Ak sa zaznamenávajú odhalené chyby každý deň, týždenná hodnota je spočítaná ako priemer, resp. medián za týždeň. Výber medzi priemerom alebo mediánom záleží na charakteristike meraných dát a riadi sa pravidlami uvedenými v predošlej podkapitole.

Grafy odchýlky resp. **rozsahu** – nazývané S-graf pre smerodajnú odchýlku a R-graf pre rozsah – je vhodné používať v kombinácií s grafmi centrálnej tendencie. Udávajú odchýlenie, s akým sa vyskytujú merané hodnoty vo vzťahu k priemeru resp. mediánu.

Kontrolné grafy (Control charts) kombinujú uvedené typy grafov spolu s limitmi. Význam kontrolných grafov spočíva v odhaľovaní odchýlok, ktoré sú zapríčinené špeciálnymi, neštandardnými udalosťami a je nutné ich preskúmať. Práve určené limity zohrávajú hlavnú úlohu v rozlišovaní špeciálnych odchýlok, od tých, ktoré nastávajú štandardne. Najdôležitejšou úlohou je teda správne určiť tieto limity. Určenie limitov sa odvíja od očakávaných hodnôt. Príklady takýchto hodnôt môžu byť: hodnoty odvodené zo skúsenosti, hodnoty vypočítané z nameraných dát, plánované hodnoty, ciele na zlepšenie a podobne. Vhodnejšie, než mať v grafe znázornenú konkrétnu očakávanú hodnotu, je mať znázornené rozmedzie hodnôt, ktoré považujeme za správne chovanie – dolný a horný limit.



Obr. 3.7: Očakávaná hodnota a limity z nej odvodené [5].

Na obrázku 3.7 vidíme dve situácie, situácia napravo znázorňuje očakávanú hodnotu v porovnaní s aktuálnou hodnotou, pričom situácia naľavo znázorňuje aktuálnu hodnotu a limity odvodené z očakávanej hodnoty (konkrétne rozdiel 3σ, pričom σ je smerodajná odchýlka). Z grafu vľavo je ťažké vyvodiť, či je správanie procesu v rámci normy, ale z grafu vpravo vidíme, že aktuálny stav sa pohybuje v rámci spodného (Lower Expectation Limit – LEL) a vrchného (Upper Expectation Limit – UEL) limitu.

Pre určenie konkrétnej hodnoty limitov môžeme použiť *Chebyshevov teorém* [5]. Ten hovorí, že pre každú dátovú sadu najmenej $(1 - (1/k^2))$ pozorovaní spadá do *k*-násobku smerodajnej odchýlky od priemeru, pričom platí, že $k \geq 1$. Tabuľkovo je teorém pre niekoľko významných hodnôt znázornený v tabuľke 3.1. na základe tohto teorému môžeme určiť spodný alebo horný limit pre kontrolný graf, podľa toho, akú veľkú presnosť vyžadujeme – koľko percent prípadov má spadať do vymedzenej oblasti.

k	1-(1/k^2)
1	0
1,5	0,56
2	0,75
2,5	0,84
3	0,89
3,5	0,92
4	0,94
5	0,96

Tab. 3.1: Niektoré významné hodnoty vypočítané podľa Chebyshevovho teorému.

Existuje viacero druhov kontrolných grafov. Medzi najčastejšie používané patria:

- X-bar graf vrchným kontrolným limitom (VKL) a spodným kontrolným limitom (SKL)
- X-bar R graf s VKL a SKL
- X-bar S graf s VKL a SKL
- XmR graf s VKL a SKL
- MMM graf
- M, σ+μ, σ-μ graf
- profil životného cyklu
- kumulatívny graf s bodovými alebo intervalovými odhadmi

Grafy typu X-bar s VKL a SKL sú vlastne grafy centrálnej tendencie s vyznačenými limitmi. Ak k takémuto grafu pridáme S-graf alebo R-graf tak dostaneme grafy typu X-bar S alebo X-bar R. Tieto grafy poskytujú okrem informácií o centrálnej tendencii grafu aj informácie o odchýlke hodnôt od centrálnej tendencie. Aj v týchto grafoch sú vyznačené VKL a SKL, aby bolo možné odhaliť neštandardné odchýlky, ktoré vyžadujú preskúmanie. Ak nie sme schopní vytvoriť graf centrálnej tendencie – jedno meranie neposkytuje skupinu hodnôt, ale iba jednu hodnotu – tak môžeme použiť graf XmR, ktorý namiesto priemeru, resp. mediánu používa priamo namerané hodnoty. Aj tento typ grafu sa skladá z dvoch grafov – graf s nameranými hodnotami a graf pre rozsah (XmR) alebo smerodajnú odchýlku (XmS).

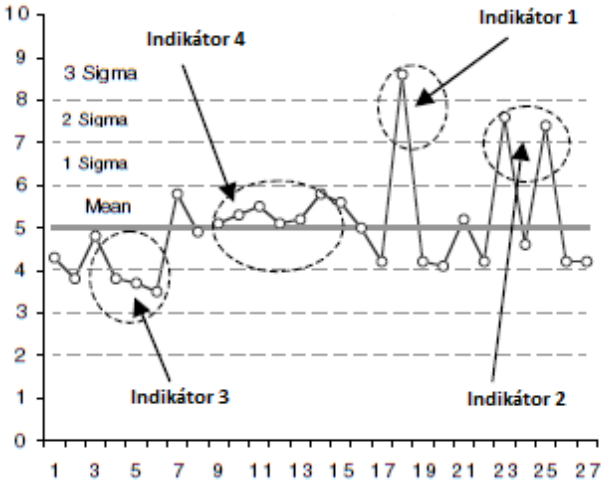
Grafy MMM znázorňujú tri hodnoty. Prvá hodnota je priemer, respektíve medián, druhá hodnota je minimum a tretia maximum. Názov MMM je odvodený z anglických názvov: mean/median, maximum, minimum. Podobný graf je graf M, σ+μ, σ-μ, kde namiesto hodnôt minima a maxima vystupujú hodnoty odvodené zo štatistických vlastností súboru dát: smerodajná odchýlka σ a stredná hodnota μ. Grafy tohto typu sa používajú na určenie výkonnostného vzoru procesu.

Grafy životného cyklu a kumulatívne grafy sa používajú na porovnanie nameraných hodnôt s očakávaniami vyplývajúcimi z plánu.

Pri vyhodnotení kontrolných grafov je vhodné sa riadiť nasledujúcimi indikátormi [5]:

1. Akýkoľvek bod vyskytujúci sa mimo limitov poukazuje na špeciálny prípad a je vhodné ho prešetriť.
2. Postupnosť viacerých nasledujúcich bodov, ktoré sú pod, resp. nad centrálnou hodnotou a majú stúpajúci resp. klesajúci charakter poukazuje na špeciálny prípad a je vhodné ho prešetriť (pozn. centrálna hodnota je priemer všetkých zaznačených hodnôt).
3. Každý neštandardný vzor, ktorý obsahuje cyklický alebo prúdový charakter, poukazuje na špeciálny prípad a je vhodné ho prešetriť.
4. Počet bodov vo vzdialenosti do jednej tretiny vzdialenosti centrálnej hodnoty a limitu by mal byť rovný približne dvom tretinám všetkých zaznačených bodov.

Jednotlivé situácie sú zachytené na obrázku 3.8.



Obr. 3.8: Znázornené situácie s vyznačenými indikátormi

V jednom kontrolnom grafe by mala byť vyznačená práve jedna metrika. Pri vyznačení viacero metrík a tým pádom aj viacero limitov by sa graf stal neprehľadným. Takisto, ak chceme do grafu vyznačiť viacero limitov pre rôzne účely (napríklad jeden pre developer, jeden pre manažéra, …), je vhodné graf zduplikovať a zakaždým vyznačiť iba potrebný limit.

Pri časovej analýze grafov je dôležité aby boli údaje usporiadané chronologicky, aby nedošlo k zlej interpretácií informácií.

Kontrolné grafy je vhodné zasadiť do kontextu. Súčasťou kontextu by mali byť informácie, na základe akých kritérií boli vybraté limity, a čo tieto limity reprezentujú (ciele, očakávané hodnoty, špecifikačné limity, …). Ďalším prvkom kontextu môžu byť aj iné grafy, ktoré napríklad znázorňujú situáciu pre body, ktoré sú mimo vyznačených limitov. Kontext kontrolného grafu môže tvoriť aj prístrojová doska procesu.

Kontrolné grafy je možné využiť pre automatickú kontrolu ako aj pre podporu rozhodovania. Pri automatickej kontrole je nutný systém, ktorý zaznamenané údaje priebežne vyhodnocuje a nachádza hodnoty, ktoré nespadajú do limitov. Ak odhalí takúto hodnotu, priamo upozorní zodpovednú osobu na problém aj s príslušnými informáciami. Týmto spôsobom nemusia byť kontrolné grafy vôbec použité ako výstup, ich použitie je vstavané v kontrole a môžu byť poskytnuté pri explicitnom vyžiadaní. Pre podporu rozhodovania je nutné aby mali kontrolné grafy istú mieru flexibility. Pri automatickej kontrole sú limity dané pevne. Zistené neštandardné situácie môžu často znamenať planý poplach. Preto pre podporu rozhodovania je nutné umožniť posúvanie limitov. Týmto spôsobom je možné na kontrolný graf nahliadať z odlišnej perspektívy a vyhodnotiť ho odlišným spôsobom.

3.3 Vzťahové vyhodnotenie

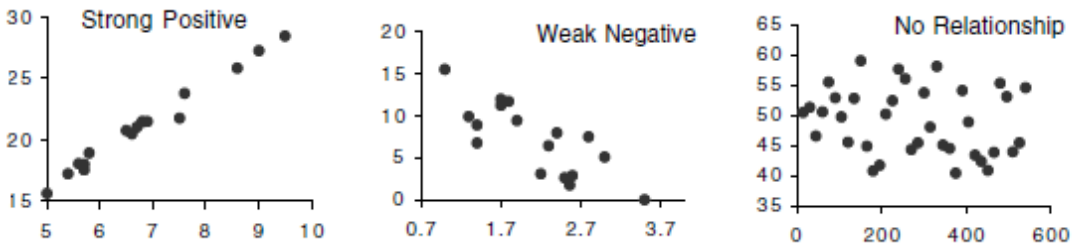
Vzťahové vyhodnotenie [5] skúma možné vzťahy medzi jednotlivými metrikami, a teda procesmi, resp. činnosťami, ktoré ich produkujú. Tvorba softwaru v sebe zahŕňa veľké množstvo procesov, ktoré sa navzájom ovplyvňujú. Analýzou metrík, ktoré tieto procesy reprezentujú, môžeme nájsť procesy, ktoré sa navzájom ovplyvňujú. Taktiež vieme určiť, do akej miery a akou formou prebieha ovplyvňovanie. Samotná informácia o tom, že spolu majú dva procesy nejakú interakciu, nemá veľké

praktické využitie. Úžitok spočíva v tom, že vieme ako sa procesy navzájom ovplyvňujú a tým pádom môžeme predvídať ich budúci vývoj. Práve odhad budúceho stavu a následné prehodnotenie využitia prostriedkov je úžitok plynúci zo vzťahového vyhodnotenia metrických údajov.

Pri zisťovaní, či existuje nejaká závislosť medzi dvoma metrikami, predstavuje ideálny prostriedok vizualizácie diagram rozptylu (scatter plot). Na x-ovej osi sa nachádza nezávislá metrika a na y-ovej osi metrika závislá. Predpokladáme, že hodnoty závislej metriky je možné vypočítať podľa nejakého vzorca z metriky nezávislej. Rozoznávame päť typov závislostí [5]:

1. silná pozitívna
2. silná negatívna
3. slabá pozitívna
4. slabá negatívna
5. žiadna

Na obrázku 3.9 sú znázornené v poradí zľava doprava silná pozitívna, slabá negatívna a žiadna závislosť. Pozitívna závislosť sa vyznačuje rastúcim charakterom z hľadiska monotónnosti, negatívna klesajúcim. Sila závislosti sa určuje na základe rozptylu bodov, čím sú body rozptýlenejšie, tým je závislosť slabšia.



Obr. 3.9: Závislosti silná pozitívna, slabá negatívna, žiadna [5].

Vzorec na výpočet sily závislosti – korelačný koeficient r – je:

$$r = \frac{\sum(\delta x)(\delta y)}{\sqrt{\sum(\delta x)^2 \sum(\delta y)^2}} \qquad (3.3)$$

Pričom $\delta x = x - priemer(x)$ a $\delta y = y - priemer(y)$. Hodnota koeficientu sa pohybuje v intervale <-1,1>, podľa toho, či sa jedná o negatívnu alebo pozitívnu závislosť. Čím väčšia je absolútna hodnota koeficientu, tým väčšia je sila závislosti.

Pri tomto spôsobe zisťovania závislostí je možné sa ľahko pomýliť. Dve metriky sa môžu tváriť ako závislé, pričom sa môže jednať o náhodu. Naopak ak je korelačný koeficient blízko nule, nemôžeme s istotou tvrdiť, že medzi metrikami závislosť neexistuje.

Vyššia úroveň určovania vzťahov medzi metrikami zahŕňa matematické vyjadrenie tohto vzťahu. Na určenie matematického vzťahu sa používa *regresná analýza*. Na znázornenie regresnej analýzy sa taktiež využívajú diagramy rozptylu, obohatené o regresné krivky, ktoré znázorňujú vypočítaný matematický vzťah. Rozoznávame niekoľko typov regresie:

- lineárna regresia
- nelineárna regresia
    - logaritmická

- polynomiálna s polynómom stupňa 2 - 4
- mocninová
- exponenciálna

Podobne ako pri určovaní sily pomocou korelačného koeficientu, je možné aj pri regresii kvantifikovať vhodnosť použitého modelu. Vhodnosť regresného modelu určuje koeficient rozhodnosti (coefficient of determination). Tento koeficient je možné vypočítať na základe vzorca:

$$R^2 = 1 - \frac{ESS}{TSS} \qquad (3.4)$$

ESS je chybová suma štvorcov (error sum of squares) a predstavuje chybu, o ktorú sa odlišuje vzorcom odhadnutá hodnota od nameranej hodnoty. Vzorec na jej výpočet je:

$$ESS = \sum(y_{nameraná} - y_{odhadnuté})^2 \qquad (3.5)$$

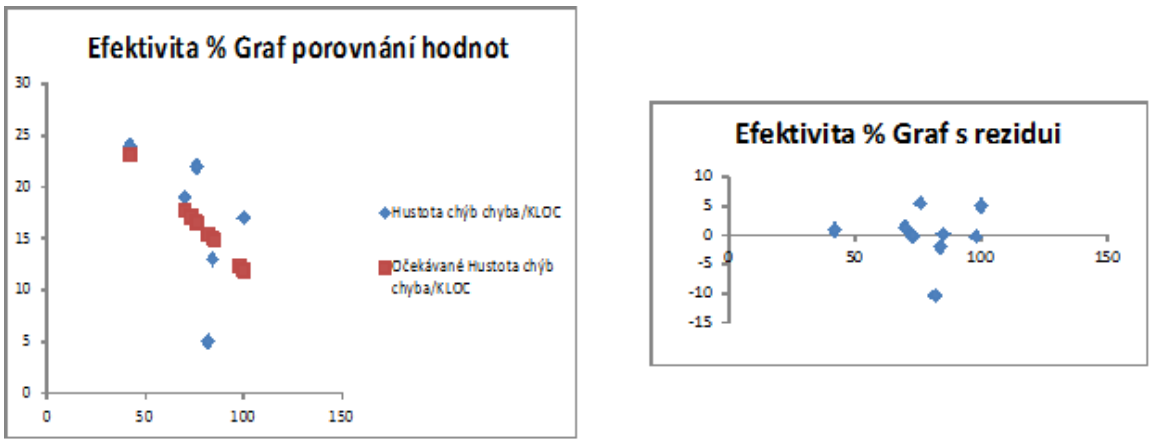
TSS je totálna suma štvorcov (total sum of squares) a udáva štvorec rozdielu medzi nameranou hodnotou a priemerom. Je vypočítaná podľa vzorca:

$$TSS = \sum(y_{nameraná} - y_{odhadnuté})^2 \qquad (3.6)$$

Okrem koeficientu rozhodnosti máme k dispozícii aj chybu strednej hodnoty (standard error of estimate – SE). Táto určuje mieru spoľahlivosti regresnej krivky ako prostriedku pre vykonanie odhadov. Vzorec na jej výpočet pre *n* vzoriek je:

$$SE = \sqrt{\frac{ESS}{n-2}} \qquad (3.7)$$

Okrem samotnej regresnej krivky je pri regresnej analýze vhodné uviesť aj doplňujúce informácie. Na obrázku 3.10 a tabuľke 3.2 až 3.4 je výber z informácií regresnej analýzy poskytovaných programom Microsoft Excel 2010. Obsiahnuté sú: graf regresnej krivky, graf rezíduí, tabuľka regresnej štatistiky, tabuľka rezíduí a zdrojová tabuľka.



Obr. 3.10: Graf regresnej krivky a graf rezíduí.

Regresná štatistika	
Násobné R	0,58353385
Hodnota spoľahlivosti R	0,340511755
Nastavená hodnota spoľahlivosti R	0,246299148
Chyba strednej hodnoty	4,930198016
Pozorovaní	9

Tab. 3.2: Regresná štatistika.

Pozorovaní	Očakávaná Hustota chýb [chyba/KLOC]	Rezíduá
1	23,10693832	0,893061684
2	17,71251526	1,287484743
3	17,13454136	-0,134541358
4	16,55656746	5,443432542
5	15,40061966	-10,40061966
6	15,01530373	-2,015303727
7	14,82264576	0,177354239
8	12,3180922	-0,318092198
9	11,93277627	5,067223735

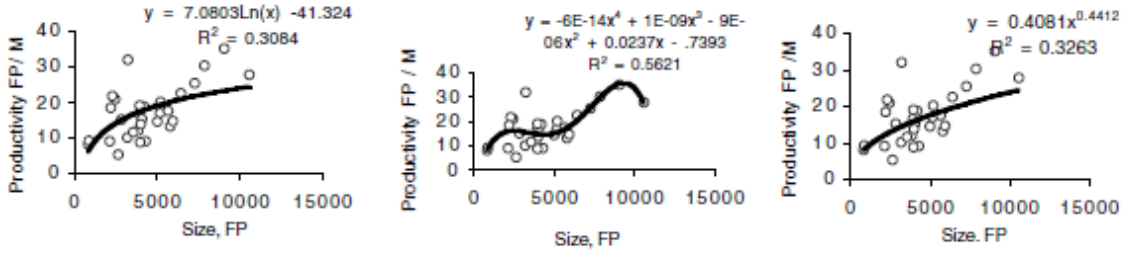
Tab. 3.3: Tabuľka rezíduí.

Efektivita [%]	Hustota chýb [chyba/KLOC]
42	24
70	19
73	17
76	22
82	5
84	13
85	15
98	12
100	17

Tab. 3.4: Zdrojová tabuľka dát.

Rezíduami rozumieme hodnoty, o ktoré sa odhadnuté hodnoty odlišujú od nameraných hodnôt. Na grafe regresnej krivky sú vyznačené namerané hodnoty a odhadnuté hodnoty vo forme postupnosti bodov. Na grafe s rezíduami sú vyznačené vzniknuté rezíduá. Regresná štatistika obsahuje koeficient rozhodnosti a jeho formy, chybu strednej hodnoty a počet meraní. Tabuľka rezíduí obsahuje odhadnuté hodnoty a vzniknuté rezíduá. Na zdrojovej tabuľke dát bola vykonaná regresná analýza.

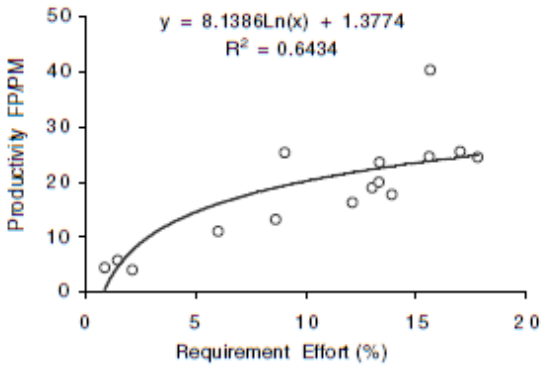
Uvedený príklad sa týka lineárnej regresie. Často sa však stretávame so vzťahmi, ktoré nie je možné charakterizovať lineárnou regresiou. V tomto prípade je vhodné použiť niektorú z nelineárnych regresií ako matematický model pre vzťah. Pri použití nelineárnej regresie ale vyvstáva problém. Samotný koeficient rozhodnosti nestačí na určenie vhodného modelu. Na obrázku 3.11 máme zobrazené tri možnosti aplikácií nelineárnych regresných modelov pre vzťah medzi veľkosťou a produktivitou.



Obr. 3.11: Grafy logaritmickej regresie, polynomiálnej 4-tého stupňa a mocninovej [5].

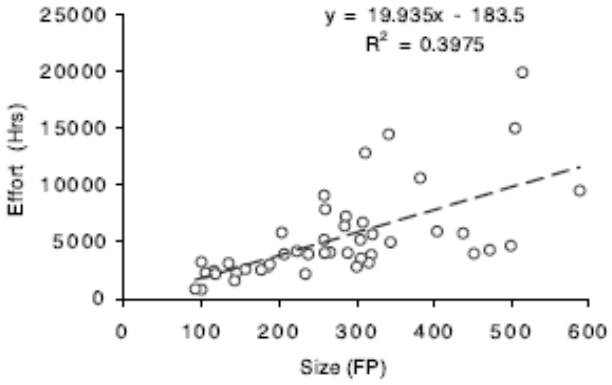
Logaritmická regresia nesie najnižšiu hodnotu koeficientu, tým pádom je najmenej vhodná pre použitie v úlohe modelu. Polynomiálna regresia má najlepšie ohodnotenie, ako model sa však nehodí. Keby sme chceli podľa polynomiálnej regresie urobiť odhad budúceho stavu, pri hodnote veľkosti okolo 15000 by sme sa dostali na zápornú hodnotu efektívnosti, čo je nezmysel. Naproti tomu mocninová regresia má síce slabšie ohodnotenie, ale jej chovanie vo väčšej miere zodpovedá očakávanému chovaniu.

Príkladom klasického využitia modelov regresie je bežná analýza dvoch množín metrických údajov. Konkrétnym prípadom môže byť vplyv úsilia vynaloženého na zber požiadaviek na produktivitu. Úsilie vynaložené na požiadavky je merané v percentách totálneho úsilia vynaloženého na projekt a produktivita ako počet jednotiek programu (function point) za jednotku úsilia (človeko/mesiac). Na obrázku 3.12 je znázornená nelineárna regresná krivka popisujúca vzťah meraných veličín. Je prirodzené, že zvýšením úsilia vynaloženého na zber požiadaviek sa zvýši produktivita. Ak sú všetky požiadavky zozbierané na začiatku projektu, nastane menej zmien v priebehu jeho vypracovania a teda aj menej komplikácií. Koeficient rozhodnosti je síce len 64.34% (ideálne prípady sa pohybujú nad 75%), ale je možné zanedbať extrémne hodnoty, ktoré sú pravdepodobne spôsobené špeciálnymi prípadmi. Na regresnej krivke vidíme, že sa postupne vyrovnáva, čo naznačuje, že zvyšovanie úsilia vynaloženého na zber požiadaviek má síce pozitívny efekt na produktivitu, ale len do istej miery. Po prekročení tejto hranice sa ďalšie zvyšovanie stáva kontraproduktívnym.



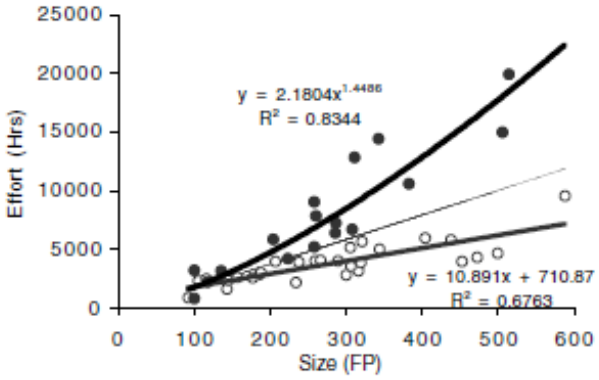
Obr. 3.12: Regresná krivka úsilia vynaloženého na zber požiadaviek a produktivity [5].

Zaujímavým prípadom využitia regresnej analýzy je vybudovanie modelu pre odhad potrebného úsilia. Uvažované metriky sú veľkosť programu (function points) a úsilie (hodiny), pričom veľkosť uvažujeme ako nezávislú premennú a úsilie ako závislú. Pri regresnej analýze bola použitá lineárna regresia (obr. 3.13). Dosiahnutá hodnota koeficientu rozhodnosti je 39,75%, čo je veľmi nízka hodnota (dobré hodnoty sú nad 75%).



Obr. 3.13: Lineárna regresia veľkosti kódu a úsilia [5].

Pri bližšom vyhodnotení grafu na obrázku 3.13 bola zistená možnosť rozdelenia meraní na dve skupiny. Jednotlivé merania predstavujú samostatné projekty. Vzhľadom na to, že softwarové projekty sa od seba často líšia zložitosťou, charakterom, alebo aj projektovým týmom, ktorý ho vypracováva, je iba prirodzené, že softwarové projekty je možné triediť do rozličných skupín. Rozdelením projektov na dve skupiny a opätovným vypracovaním analýzy pre tieto skupiny samostatne došlo k výraznému zlepšeniu koeficientu rozhodnosti na 83,44% a 67,63%.



Obr. 3.13: Lineárna regresia veľkosti kódu a úsilia po rozdelení na dve skupiny [5].

4 Špecifikácia požiadaviek a návrh architektúry systému

Táto kapitola a všetky nasledujúce sa venujú samotnému systému pre podporu metrík pri vývoji softwaru. Sú v nich uvedené a odôvodnené myšlienkové pochody, ktoré viedli k špecifikácií, návrhu a implementácií. Námety na špecifikáciu boli čerpané z literatúry a skúseností získaných pri vypracovávaní školských projektov. Návrh a implementácia boli už založené zväčša na získaných vedomostiach.

4.1 Špecifikácia požiadaviek na systém pre podporu metrík

Prvý krok pri špecifikácií požiadaviek na systém pre podporu metrík v projektoch vývoja softwaru, je určiť, na čo bude tento systém slúžiť. Metriky sa zavádzajú za účelom sledovania postupu, ktorým sa dosahujú ciele organizácie. Pomocou metrík sme schopný vyjadriť súčasný stav a odhadnúť, koľko zdrojov nám chýba na dosiahnutie cieľa. Metriky sú teda zamerané na ciele organizácie. Systém, ktorý metriky spravuje, musí byť teda tiež zameraný na ciele organizácie. Musí zhromažďovať všetky nazbierané informácie, vedieť ich znázorniť a aj poslať ďalej. Úlohu systému pre podporu metrík môžeme teda špecifikovať pomocou nasledujúcich bodov:

- získavanie metrických dát
- uchovávanie metrických dát
- sprostredkovanie získaných informácií a dát

Získavanie metrických dát je prvým krokom v aplikácií metrík. Najprv je potrebné dáta získať – namerať. V súčasnosti existuje veľké množstvo programov a prostriedkov ktoré umožňujú získavať metrické informácie. Každý z týchto prostriedkov je zameraný na špecifický predmet, resp. skupinu predmetov, ktorých atribúty meria. Spoločné pomenovanie pre takéto programy je CAME-nástroje. Medzi najznámejšie patria RSM (analýza dokumentácie a zdrojových kódov), MetricsOne (analýza UML diagramov). Tieto nástroje často okrem samotného zberu dát poskytujú aj analýzu zozbieraných dát a prezentáciu výsledkov vo forme tabuliek diagramov a podobne. Vzhľadom na existenciu veľkého množstva špecializovaných prostriedkov sa problematika získavania metrických dát mení na problematiku importu získaných dát. Ak chceme aby jeden systém spravoval všetky získané metriky, musíme zabezpečiť rýchly, jednoduchý a hlavne automatizovaný. Využitie existujúcich prostriedkov síce rieši problém zberu informácií, ale na druhú stranu prináša problém zložitého rozhrania. Programy na zber metrík môžu získané dáta prezentovať odlišnými spôsobmi. Najhoršou možnosťou je, že program zozbiera dáta a poskytne ich iba vo forme grafov, alebo tabuliek, ktoré sa nedajú exportovať a zaniknú pri zastavení programu. Takéto programy sú ako vstup pre systém nepoužiteľné. Ideálne rozhranie ponúkajú programy, ktoré vyvážajú získané dáta vo forme textového súboru, obsahujúceho hodnoty oddelené oddeľovačom. Je pravdepodobné, že súbor bude obsahovať dodatočné informácie o dátach, a že použitý oddeľovač sa bude líšiť medzi jednotlivými nástrojmi. Tieto problémy sú však ľahko riešiteľné a miera automatizácie importu daného typu dát je veľká. Ďalšia možnosť je, že programy exportujú dáta v špecifickej štruktúre – napríklad vo formáte xml, alebo html. Pri tejto forme dát je nutné disponovať prostriedkami na parsovanie údajov. Okrem dát importovaných z meracích programov je nutné umožniť aj manuálne vkladanie dát. Vzhľadom na charakter metrických dát je možné predpokladať, že manuálne vložené dáta budú mať taktiež formu

štruktúrovaného súboru. Najčastejšie sa môže jednať o textový dokument s dátami oddelenými cez oddeľovače, alebo o tabuľkové súbory, ktoré sú produktom nástrojov ako Microsoft Excel.

Uchovávanie metrických dát implikuje existenciu databázy, do ktorej budeme metrické dáta ukladať a spravovať ich. V prvom rade je nutné určiť, ktoré informácie o meraní a metrike je nutné ukladať, a ktoré nie. Požadované informácie pre metriky môžu byť:

- názov
- účel
- definícia
- stupnica
- predpokladaný rozsah hodnôt
- hraničné limity
- zdroj dát
- formát vstupných metrických dát
- frekvencia zberu
- metódy analýzy
- cena metriky

Každá metrika musí mať určený názov, aby ju bolo možné identifikovať. Vzhľadom na to, že metriky sú zamerané na ciele je vhodné pri každej uviesť, za akým účelom je zaznamenávaná. Ak by bola definovaná nejaká metrika, ktorá nemá jasný účel, bolo by zbytočné ju zaznamenávať. Naopak existuje možnosť, že metrika má svoj špecifický význam, ale cieľ, ktorý sleduje nebol identifikovaný. Aby sme predišli týmto prípadom, je nutné pri metrike uvádzať jej účel.

Definícia metriky obsahuje presný popis jej získania. Nie je podstatné, či je metrika získavaná manuálne alebo pomocou nejakého programu, je dôležité mať uvedený presný postup získania údajov – meranie, alebo výpočet.

Pri metrikách je nutné uviesť v akej stupnici je definovaná. Z tejto informácie vyplývajú operácie, ktoré môžeme s nameranými dátami vykonávať. Okrem stupnice je vhodné uvádzať aj predpokladané rozsahy hodnôt. Rozsahy sú dôležité nielen z implementačného hľadiska, ale aj z hľadiska analýzy údajov – prítomnosť nepredvídanej hodnoty môže znamenať problém. Podobne ako rozsahy hodnôt je vhodné uviesť aj hraničné limity. Tie predstavujú hodnoty očakávané pri normálnom fungovaní monitorovanej činnosti. Presiahnutie limitu indikuje problém. Zavedenie limitov taktiež podmieňuje implementáciu automatizovaného systému upozorňovania. Je ale nutné brať do úvahy existenciu viacerých limitov a možnosť zmeny týchto limitov. Taktiež sa môžu jednotlivé limity meniť na základe projektu, na ktorý je metrika aplikovaná.

Zdroj dát a formát vstupných dát sú dôležité z hľadiska prvej spomínanej úlohy systému – získavania dát. Predpokladá sa, že organizácia má jednotný systém získavania dát. To znamená, že na získanie jednej metriky sa používa práve jeden prostriedok. Zmena prostriedku často znamená aj zmenu formátu a tá podmieňuje aj zmenu rozhrania systému, ktoré umožňuje import metrických dát.

Frekvencia zberu metriky je jedným z najdôležitejších údajov. Udáva interval, v ktorom je metrika zaznamenávaná. Pre správne fungovanie systému metrík je nutné dodržiavanie frekvencie zberu dát. Nekonzistentné údaje sú znehodnotené a neposkytujú vhodnú základňu pre rozhodovanie – často totiž vedú k nesprávnym záverom.

Okrem reprezentácie samotných metrických dát býva od metrickej databáze často vyžadované aj nasledujúce [2]:

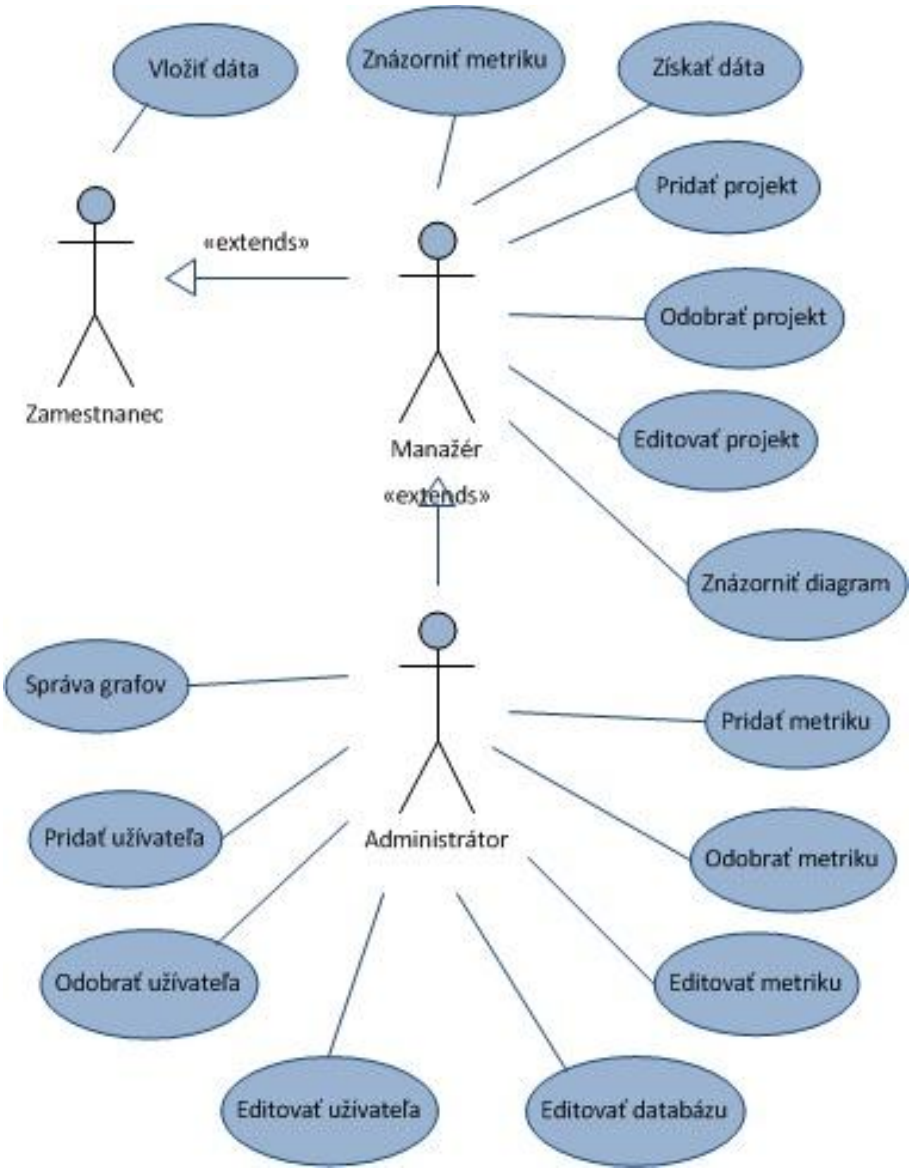
- úsilie vynaložené na používanie a správu databázy, resp. systému by malo byť minimálne, čo podmieňuje veľkú mieru automatizácie

- je nutné, aby boli metrické údaje spájané s projektom, na ktorom boli vykonané merania a taktiež s jednotlivými artefaktmi (zdrojové kódy, diagramy, …), prípadne s procesmi
- je nutná diferenciácia prístupových práv – administrátor na manipuláciu s databázou, manažér na zobrazovanie údajov a diagramov, …
- mala by byť zahrnutá automatická detekcia problémov – zistenie prekročenia limitov nastavených na metriku a automatické informovanie užívateľa, súčasne existencia viacerých limitov
- existencia možnosti zobrazovania informácií pomocou grafov a porovnávanie týchto informácií medzi rôznymi projektmi
- podpora dlhodobého zaznamenávania pre použitie v modeloch pre odhad úsilia, nákladov, produktivity
- systém musí podporovať pridávanie nových metrík
- systém by mal podporovať pridávanie nových foriem vyhodnotenia, ktoré nie sú zahrnuté, napríklad pomocou add-onov
- mal by byť možný export údajov a vykonaných analýz vo vhodnom formáte
- jednoduchý prístup pomocou intranetu

Sprostredkovanie získaných informácií a dát sa týka exportu a znázornenia dát. Podobne ako prostriedky na zber metrík poskytujú získané informácie v nejakom formáte pre ďalšie spracovanie, aj systém pre podporu metrík by mal byť schopný poskytnúť zhromaždené informácie. Exportované informácie bude možné využiť v správach a v dokumentácii. Vzhľadom na to, že dokumentácia a správy sú tvorené ľuďmi pre ľudí, je vhodné pre export zvoliť priateľský užívateľský formát. Príkladom takéhoto formátu je výstup spracovateľný pomocou programu Microsoft Excel. Vhodná by bola aj podpora exportu grafických informácií – diagramov – a podpora clipboardu. Inou formou následného využitia je pokročilá analýza dát. Implementovať všetky druhy analýzy do samotného systému by bolo časovo veľmi náročné. Zároveň by sa zvýšila zložitosť systému a tým pádom jeho náchylnosť k chybám, ktorá musí byť držaná na minime. Preto je vhodné na pokročilú analýzu použiť osobitné aplikácie. Takéto využitie predpokladá použitie formátu, ktorý je ľahko spracovateľný automatickou formou.

V predošlých odsekoch boli často spomínané grafické informácie, hlavne grafy. Jednou zo základných funkcií systému pre podporu metrík by mala byť schopnosť reprezentovať metrické informácie graficky. Tabuľková forma býva pre ľudského užívateľa často neprívetivá a je v nej veľmi zložité spozorovať existenciu vzoru. V grafickej reprezentácií je možné odhaliť vzory, alebo špeciálne prípady ľudským užívateľom podstatne jednoduchšie, čo je dôvod na zavedenie grafickej reprezentácie. V kapitole 3 boli popísané možné analýzy metrických údajov a grafické reprezentácie používané k týmto analýzam. Bolo by vhodné implementovať všetky uvedené reprezentácie, konkrétne: histogram, distribučná krivka, kontrolný graf, graf rozptylu, regresná krivka, graf rezíduí. Okrem uvedených foriem reprezentácie je možné systém obohatiť o niektoré ďalšie reprezentácie, ktoré majú využitie hlavne pri prezeraní dát na projektovej úrovni, resp. na úrovni manažmentu: koláčový graf, radarový graf, ručičkový ukazovateľ (forma tachometru).

Na základe uvedených požiadaviek bol vytvorený diagram prípadov použitia (Use case) znázornený na obrázku 4.1.

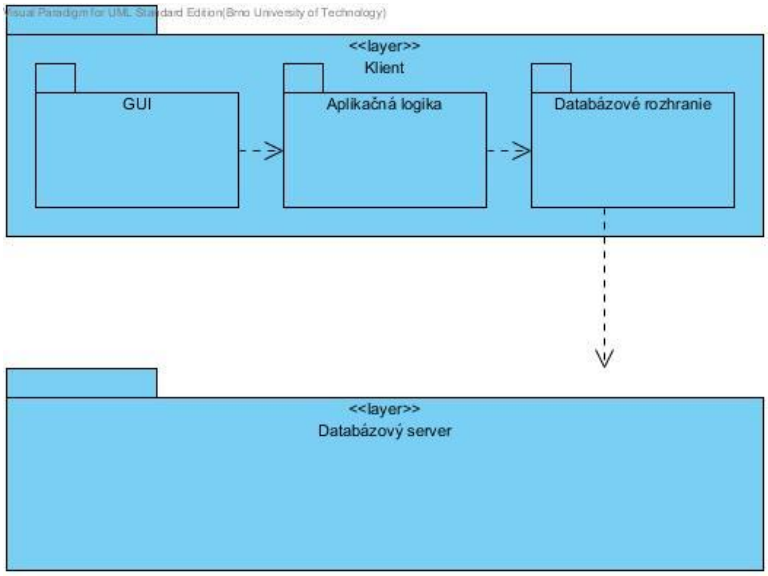


Obr. 4.1: Diagram prípadov použitia pre systém pre podporu metrík.

4.2 Návrh architektúry systému pre podporu metrík

Voľba architektúry, ktorá bude použitá pre systém pre podporu metrík musí byť založená na požiadavkách na systém. Jedna z možností je použiť architektúru s dvomi hlavnými vrstvami: „tučný“ klient a databázový server (obr. 4.2). V takejto architektúre klient obsahuje rozhranie, kompletnú aplikačnú logiku a aj rozhranie pre komunikáciu s databázou.

Výhodou tejto architektúry je, že všetky výpočty a operácie sú vykonávané na klientovi – tým pádom nie je potrebná existencia serveru, ktorý by musel obstarávať náročné operácie. Ďalšou výhodou je jednoduchý návrh a implementácia. Všetky súčasti sa nachádzajú v rámci jednej aplikácie. Samotný systém by bol potom tvorený sieťou klientov a samotným databázovým serverom.



Obr. 4.2: Dvojvrstvová architektúra klient a databázový server.

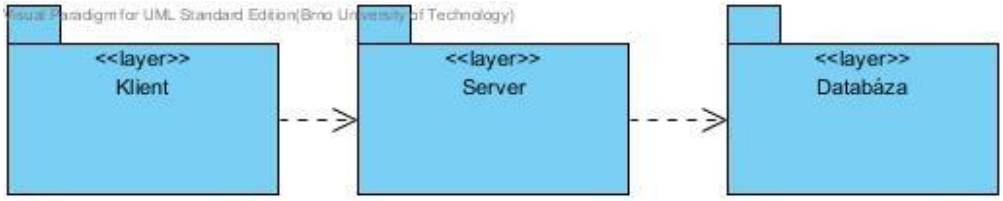
Nevýhody, ktoré vyplývajú z uvedenej architektúry však vážne prevažujú nad výhodami. Samotný klient bude mať relatívne vysoké výpočetné nároky. Všetci užívatelia používajú rovnakú verziu klienta. Nie všetci užívatelia majú však práva na využitie všetkých funkcií obsiahnutých v klientovi. To znamená, že pre rôznych užívateľov nebude využitý plný potenciál aplikácie a tým pádom nastane plytvanie vyhradenými zdrojmi. Veľmi významný problém predstavuje synchronizácia údajov. Pri zmene údajov jedným klientom sa už načítané údaje ostatných klientov stávajú zastaranými. To môže viesť k nepresným záverom, ktoré boli vyvodené z neaktuálnych dát. Ďalej je nutná osobitá inštalácia instancie klienta na každé zariadenie, na ktorom sa bude systém používať.

Lepšie riešenie predstavuje trojvrstvová architektúra klient-server-databáza (obrázok 4.3). Pri použití takejto architektúry prebieha na klientovi iba zobrazovanie výstupov a tvorba požiadaviek na server. Hlavná aplikačná logika systému a tým pádom väčšina výpočetného zaťaženia, sa nachádza na serveri. Požiadavky vysielané klientom zachytáva server. Ten ich spracuje: získa potrebné údaje z databázy, vypočíta požadované výstupy; a výsledok zašle späť klientovi.

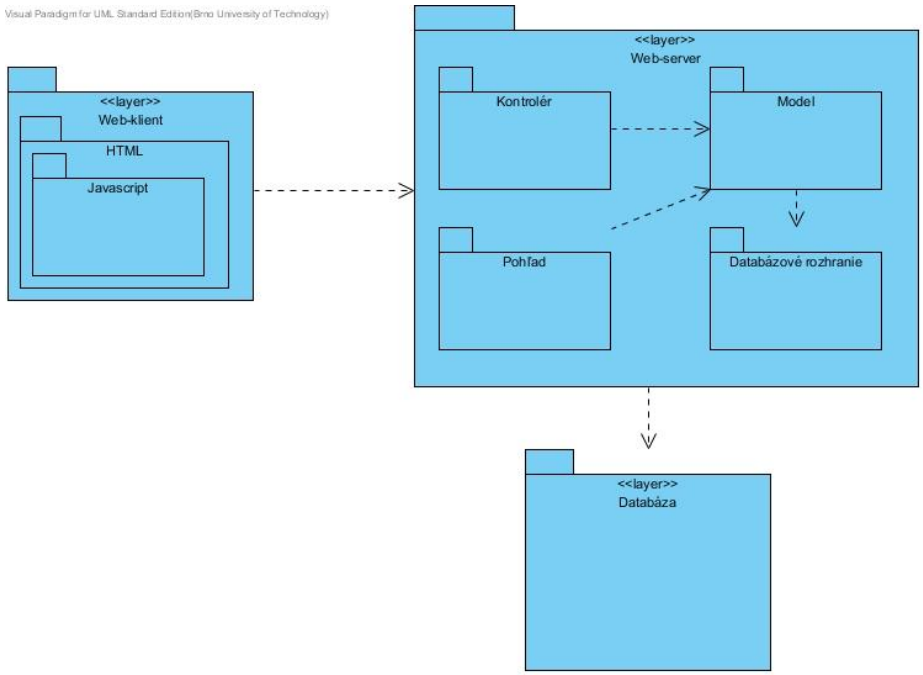
Hlavná výhoda uvedenej architektúry je prenos výpočtov na server. Vzniká tým síce potreba existencie serveru, na ktorom bude serverová vrstva architektúry nasadená, ale na druhú stranu je výrazne odľahčený klient. Takýto klient nemá vysoké požiadavky na pracovnú stanicu, na ktorej bude nasadený. Ďalej je odstránený problém nekonzistencie dát. Všetka manipulácia s dátami a výpočet operácií nastáva prostredníctvom serveru. Klienti teda nepracujú s lokálnymi kópiami dát, akurát s výstupmi operácií. Preto má klient k dispozícii vždy najaktuálnejší výstup.

Istú nevýhodu predstavuje pretrvávajúca potreba inštalácie lokálnej kópie klienta na každú pracovnú stanicu. Iná nevýhoda vyplýva z požiadaviek na systém. Jedným z uvedených bodov je jednoduchý prístup pomocou intranetu. Vzhľadom na to, že klient stále predstavuje samostatnú aplikáciu, je integrácia s intranetom náročná.

Odstránenie spomenutých nevýhod je možné dosiahnuť využitím správnych prostriedkov pre implementáciu uvedenej architektúry – webová aplikácia. Ak bude systém navrhnutý a implementovaný ako webová aplikácia, bude klient zredukovaný na internetový prehliadač. Tým pádom zaniknú problémy s lokálnou kópiou klienta (každý operačný systém zahŕňa internetový prehliadač), ako aj problémy s prenositeľnosťou. Integrácia do intranetu bude tiež jednoduchá – prostredníctvom hypertextového odkazu. Výsledná architektúra je znázornená na obrázku 4.4.



Obr. 4.3: Architektúra klient-server-databáza.



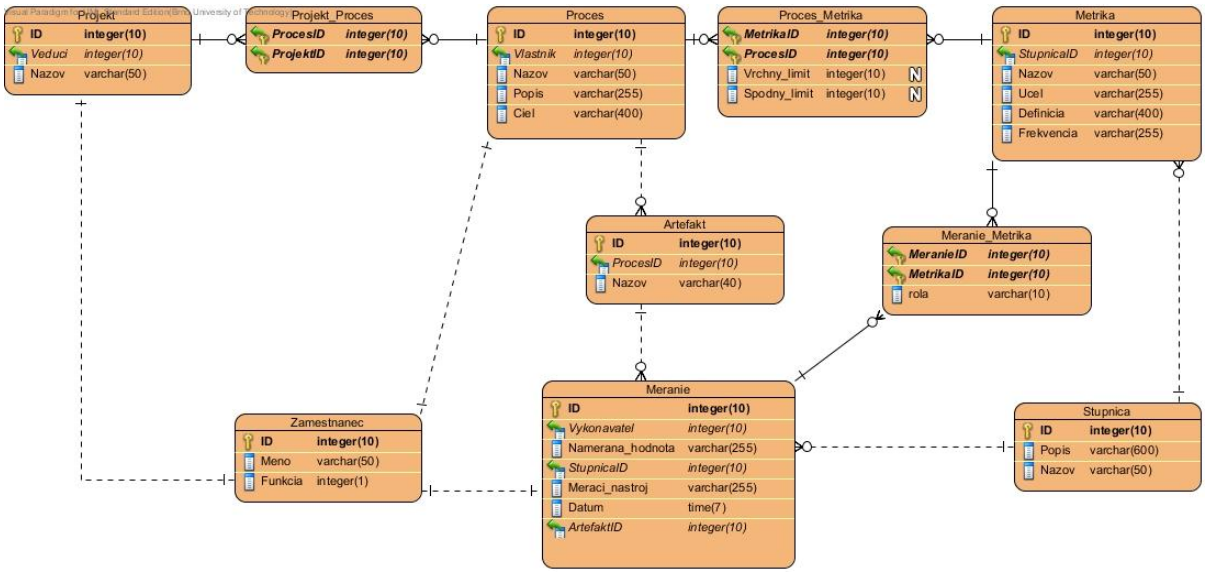
Obr. 4.4: Webová architektúra klient-server-databáza.

Základnou časťou architektúry je databáza. Tá musí byť vhodne navrhnutá, aby bolo možné systém aplikovať v reálnych podmienkach. V súčasnosti je rozšírené procesné riadenie projektov. To znamená, že projekt je definovaný ako postupnosť procesov. Každý proces má určený vstup, výstup a činnosti, ktoré sú v rámci procesu vykonávané. Procesné riadenie je aj súčasťou štandardov a metodológií. Napríklad metodológia *Project Management Body Of Knowledge* (PMBOK) definuje jednotlivé procesy a ich kategorizáciu pre obecný projekt. S úpravami je možné PMBOK aplikovať aj na softwarové projekty. PMBOK rozlišuje dve skupiny procesov: procesy projektového manažmentu a produktovo-orientované procesy. Procesy orientované na produkt bývajú súčasťou životného cyklu projektu. Štandard, ktorý definuje procesy pre životné cykly softwarových projektov je ISO 12207.

Kvôli rozšírenosti procesného riadenia, výuke zameranej na procesné riadenie a existencii metodológií a štandardov pre podporu procesného riadenia, je v návrhu databázy uvažované práve procesné riadenie projektu. ER diagram s počiatočným návrhom databázy pre systém pre podporu metrík je znázornený na obr. 4.5. Centrálnym prvkom databázy je *proces*. Z viacerých procesov je zložený *projekt*. Jednotlivé projekty sa od seba môžu odlišovať, preto je nutné zohľadniť, že ani procesné zloženie projektov nebude vždy rovnaké. Na túto štruktúru sú aplikované *metriky*. Jednotlivé metriky sa viažu na procesy, aby bolo možné procesy, ako aj projekty porovnávať navzájom. Atribúty metrík boli zvolené na základe požiadaviek uvedených v predošlej podkapitole. Nie všetky požiadavky sú obsiahnuté v rámci atribútov metriky. *Stupnica*, v ktorej je metrika reprezentovaná, bola osamostatnená ako entita. Vzhľadom na dvojaký charakter metrík – priame alebo vypočítané – dáta, použité k určeniu hodnoty metriky sú reprezentované pomocou osobitnej entity *meranie*. Z nameraných dát je potom nožné vypočítať hodnoty požadovanej metriky. Postup

výpočtu metriky je obsiahnutý v atribúte *definícia*. Vzhľadom na požiadavku maximálnej automatizácie a možnosti manuálneho pridávania metrík do systému, je nutné, aby definíciu výpočtu metriky bolo možné analyzovať aj algoritmicky. Tým pádom musí mať tento atribút formu vzorca, ktorý je možný analyzovať (napr. pomocou zásobníkového automatu). Samotné merania sú vykonávané na *artefaktoch* procesu. Pod pojmom artefakt procesu sa rozumieme činnosť, alebo výstup činnosti. Poslednou entitou v diagrame je *zamestnanec*, ktorý zohráva úlohu pri meraní, kde je nutné uvádzať osobu, ktorá meranie vykonala a pri projekte a procese.

Pri návrhu databázy došlo ku konfliktu medzi požiadavkou na minimálne úsilie na údržbu systému a potrebnou úrovňou detailov. Z hľadiska minimálneho úsilia je vhodné, aby databáza obsahovala čo najmenší počet entít a záznamov. Na druhú stranu je nutné, aby bola štruktúra projektu modelovaná na primeranej úrovni abstrakcie. Konkrétny prípad konfliktu je dvojica entít *proces* a *artefakt*. Proces je postupnosť činností, ktoré sa skladajú z jednotlivých úloh a poskytujú výstupy na základe dodaných vstupov. Podľa takejto definície procesu by bolo nutné zahrnúť tri, alebo štyri entity do návrhu databázy. Na druhú stranu takéto množstvo entít by znamenalo väčšie úsilie vynaložené na správu databázy a možnú redundanciu dát, ak firma využíva aj iné prostriedky pre správu projektu, ktoré obsahujú projektovú databázu. Miesto toho sú v návrhu uvedené len entity pre *proces* a *artefakt*, pričom pod artefaktom chápeme činnosť, výstup, resp. úlohu. Správnosť navrhnutej architektúry je možné určiť len postupom času pri implementácii a prípadné nedostatky odstrániť. Preto bude potrebné udržiavať vysokú mieru udržovateľnosti aplikácie.

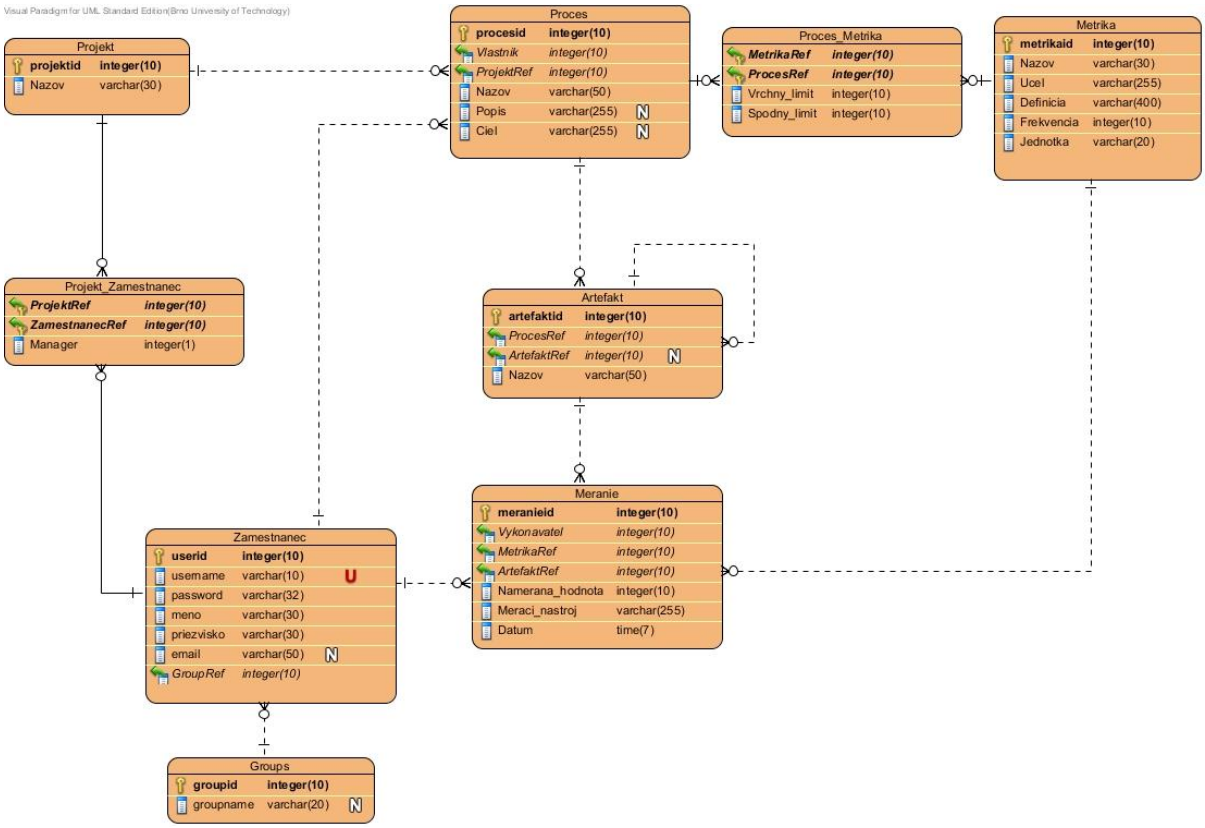


Obr. 4.5: Počiatočný ER diagram databáze pre systém pre podporu metrík.

4.3 Spresnenie návrhu systému

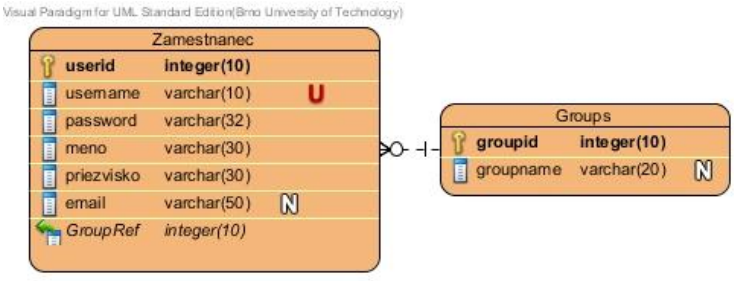
Počas bližšej analýzy návrhu databázy pre systém pre podporu metrík došlo k niekoľkým zmenám. Finálny ER diagram je znázornený na obrázku 4.6. Došlo k viacerým zmenám v atribútoch jednotlivých entít, ktoré budú spolu s významom entít popísané v ďalších odsekoch. Jednou z výraznejších zmien je odstránenie entity *Stupnica*. Podľa kapitoly 2.2 je stupnica jedným z dôležitých prvkov merania a metrík – je to forma reprezentácie nameraných hodnôt. Jednotlivé metriky pre projekty môžu byť prirodzene vyjadrené v rôznych stupniciach (ako je tiež uvedené v kapitole 2.2), tento fakt je nutné rešpektovať. Otázne je, či je nutné zakomponovať podporu pre rozličné druhy stupníc. Môžeme nadefinovať nekonečne mnoho stupníc s rôznymi hodnotami, ktoré spadajú do rozdelenia v kapitole 2.2. Výhodnejšie by však bolo, vybrať si jednu stupnicu vysoko postavenú v rozdelení, ktorá pokrýva vlastnosti nižších stupníc. Pre túto úlohu bola zvolená stupnica

celých čísel, implementačne reprezentovaná typom integer. Veľké množstvo obecne používaných metrík je možné reprezentovať pomocou celých čísel (veľkosť, počet chýb ...). Aj metriky, ktoré sú vyjadrené vo forme indexu (napr. dodržiavanie plánu – schedule compliance) je možné vyjadriť pomocou percentuálnej hodnoty – teda celým číslom. Pri metrikách, ktorých namerané hodnoty často obsahujú desatinnú časť (výsledok delenia, odmocnín) dôjde k zaokrúhleniu hodnoty a tým pádom k strate presnosti, avšak zaznamenané údaje majú stále zmysel. V nasledujúcej časti budú popísané jednotlivé entity a ich význam.



Obr. 4.6: Finálny ER-diagram databázy pre systém pre podporu metrík.

Zamestnanec je entita, ktorá reprezentuje jednotlivých užívateľov aplikácie, resp. ako názov napovedá zamestnancov spoločnosti. Entita je identifikovaná primárnym kľúčom *userid*, ktorý má hlavne význam v implementačnom kontexte. Atribúty *username* a *password* sú použité na autentizáciu užívateľa pri vstupe do aplikácie. Atribúty *meno, priezvisko* a *email* slúžia ako identifikácia užívateľa v rámci aplikácie a teda v rozhraní aplikácie sú používané práve tieto. Neoddeliteľnou súčasťou entity *Zamestnanec* je entita *Groups*. Táto entita nachádza využitie hlavne pri autentizácií užívateľa. Každý zamestnanec musí patriť práve do jednej skupiny užívateľov. Aplikácia podporuje 3 skupiny užívateľov (ako naznačuje Use-case diagram na obr. 4.1) – user, manager, admin. Počet záznamov je pevne daný a nemenný počas behu aplikácie, takisto ako sú pevne dané hodnoty záznamov tabuľky implementujúcej entitu *Groups* pri vytvorení databázy.



Obr. 4.7: Entity Zamestnanec a Groups.

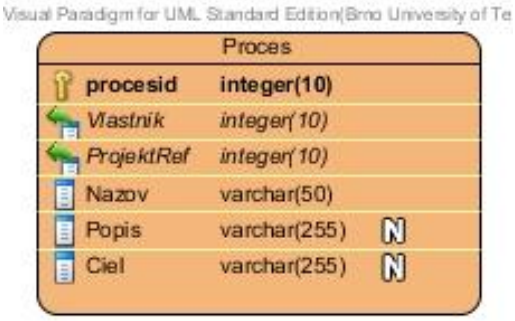
Projekt je entita reprezentujúca jednotlivé projekty, ktoré sú merané a vyhodnocované. Atribút *projektid* je unikátny identifikátor, ktorý má význam hlavne z hľadiska implementácie. Samotný projekt je v aplikácii reprezentovaný atribútom *Nazov*.

Obr. 4.8: Entita Projekt.

Projekt_Zamestnanec je entita, ktorá spája jednotlivé projekty s užívateľmi. Každý projekt má priradenú množinu ľudí, ktorí na ňom pracujú. Toto priradenie je uložené práve v entite *Projekt_Zamestnanec*. Primárny kľúč entity je zložený z dvoch atribútov, *ProjektRef* a *ZamestnanecRef*, ktoré odkazujú na jednotlivé instancie vo vzťahu. Atribút *Manager* určuje, či je daný užívateľ zároveň aj manažérom projektu. Manažérom projektu sa môže stať jedine užívateľ zo skupiny manager. Zároveň každý projekt má práve jedného manažéra. Tento má potom právo spravovať celý projekt – pridávať a odoberať užívateľov pracujúcich na projekte, editovať atribúty, procesy a podobne.

Obr. 4.9: Entita Projekt_Zamestnanec.

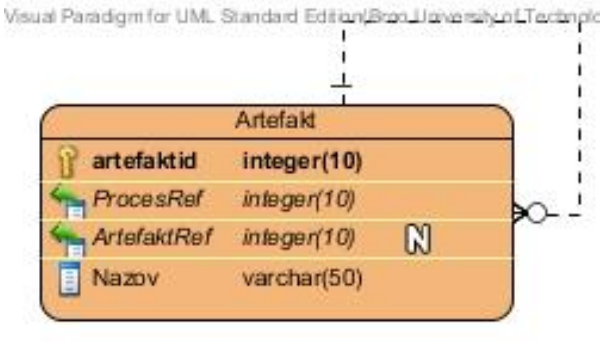
Proces reprezentuje jednotlivé procesy v rámci projektu. Proces by mal mať jasne definovaný názov (*Nazov*), popis (*Popis*) a cieľ (*Ciel*). Pre aplikáciu je kritický hlavne názov procesu, popis a cieľ nemusia byť uvedené, čo sa týka implementácie. *Proces* je jednoznačne identifikovaný atribútom *procesid*. Každý proces má svojho vlastníka (*Vlastnik*). Tento vlastník môže byť len užívateľ, ktorý je priradený k projektu, ktorého súčasťou daný proces je. Toto obmedzenie je implementované na aplikačnej úrovni, nie na databázovej. *Proces* je jedna z hlavných entít aplikácie. Práve porovnávanie procesov a získavanie informácie o procesoch je hlavnou úlohou aplikácie.



Obr. 4.10: Entita Proces.

Artefakt je špeciálna entita, ktorá v rámci aplikácie zohráva viacero úloh. Proces, ako bolo uvedené v predošlej kapitole, je postupnosť činností, aktivít a ich výstupov. Entita proces reprezentuje práve činnosti a ich výstupy. V pôvodnom návrhu sa uvažovalo, že činnosti, aktivity a ich výstupy budú reprezentované jedinou entitou – *Artefakt*. Takáto forma reprezentácie je síce jednoduchá z hľadiska

návrhu a implementácie, ale strácame pri nej podstatnú časť kontextu. Ak by sme činnosti, aktivity a ich výstupy reprezentovali ako samostatné záznamy, neuchovala by sa nám informácia, ktorý výstup patrí ktorej činnosti, resp. aktivite. Zachovanie tohto kontextu sa dosiahlo pridaním unárneho vzťahu na entitu *Artefakt*. Pre zjednodušenie neuvažujeme v procese činnosti, ale len aktivity a ich výstupy. Atribút *ArtefaktRef* predstavuje spomenutý unárny vzťah. Ak daný artefakt má hodnotu atribútu *ArtefaktRef* prázdnu (resp. rovnú *NULL*), tak sa jedná o činnosť. Ak sa atribút *ArtefaktRef* odkazuje na iný záznam v tabuľke, jedná sa o výstup (produkt) činnosti, na ktorú tento atribút odkazuje. Týmto spôsobom bolo možné zachovať kontext medzi činnosťami a ich výstupmi bez značného zvýšenia zložitosti návrhu. Samotná entita je v aplikácií reprezentovaná svojím názvom (*Nazov*) a jednoznačne priradená k procesu pomocou atribútu *ProcesRef*.



Obr. 4.11: Entita Artefakt.

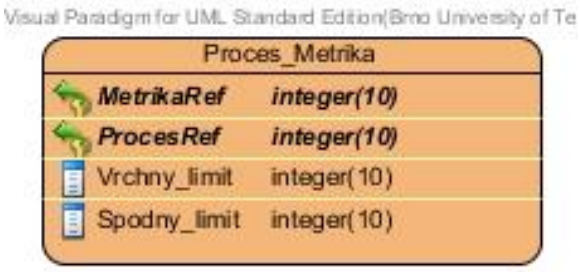
Metrika je entita reprezentujúca samotné metriky. Jednoznačný identifikátor metriky je atribút *metrikaid*. Pri metrikách je nutné pamätať si ich názov (*Nazov*), účel (*Ucel*), definíciu (*Definicia*), frekvenciu meraní (*Frekvencia*) a jednotku (*Jednotka*), v ktorej je metrika meraná. V pôvodnom návrhu sa uvažovalo, že atribút *Definicia* bude obsahovať vzorec na výpočet metriky z údajov už uložených v databáze. Od tohto prístupu sa upustilo. Definícia obsahuje postup, ako získať údaje (postup merania), čo môže zahŕňať prípadne aj vzorce na výpočet. Avšak aplikácia nepodporuje automatický výpočet hodnôt metriky z hodnôt meraní iných metrík už uložených v databáze. Hlavným dôvodom je nedostatok kontextu. Ak by sme chceli automatizovať počítanie odvodených metrík, museli by sme uchovávať informácie o kontexte – ktoré merania jednotlivých metrík sa majú použiť na výpočet hodnoty pre zvolenú metriku. To znamená, že každému jednému vloženému meraniu by sa tento kontext musel priradiť. Príkladom by mohli byť metriky veľkosť kódu, efektívnosť a hustota chýb. Uvažujme veľkosť ako počet riadkov kódu, efektívnosť ako počet riadkov kódu napísanú za jednotku času a hustotu chýb ako počet chýb na riadok kódu. Metrika veľkosť by bola použitá na automatický výpočet hodnôt pre metriky efektívnosť a hustota chýb. Preto by bolo nutné každé meranie veľkosti na každom produkte činnosti spojiť s odpovedajúcim meraním času, resp. počtu chýb. Je otázne, či je možné túto operáciu bezchybne automatizovať – museli by sa riešiť problémy s nesúladom času, neexistenciou meraní a podobne. Ak by túto operáciu automatizovať nešlo, predstavovala by vysokú záťaž na zadávateľa hodnôt meraní – pre každú jednu operáciu vloženia merania by musel vykonať ďalšie dve operácie priradenia kontextu. Okrem toho by sa výrazne zvýšili aj nároky na úložný priestor – každý záznam merania by tým pádom niesol ďalšie 2 záznamy o kontexte. Kvôli týmto dôvodom aplikácia neumožňuje automatický výpočet odvodených metrík.

Atribút *Frekvencia* určuje frekvenciu, v ktorej by mali byť vykonávané merania. Jednotka tohto atribútu je deň. Oproti pôvodnému návrhu pribudol atribút *Jednotka*. Prítomnosť tohto atribútu vyplýva z odstránenia entity *Stupnica*. Pôvodne boli jednotky merania vedené v entite *Stupnica*. Pri odstránení tejto entity musel prejsť tento atribút k entite *Metrika*. Druhou možnosťou bolo preniesť atribút *Jednotka* na entitu *Merania*. Toto riešenie je však nevhodné. Je dôležité aby metriky boli konzistentné naprieč jednotlivými projektmi, procesmi a ich artefaktmi. Ak by bola *Jednotka* atribútom merania, mohlo by sa ľahko stať, že merania na jednej činnosti (produkte) v rámci procesu

by mali odlišnú jednotku od druhej činnosti (produktu). Tým pádom by nebolo možné tieto činnosti (produkty) medzi sebou porovnať ani agregovať namerané hodnoty do zobrazenia priebehu procesu.

Obr. 4.12: Entita Metrika.

Proces_Metrika je entita, ktorá priraďuje jednotlivým procesom metriky. Manažér si určí, ktoré metriky sa budú na konkrétnom procese merať. Potom môžu byťmerania, vykonané na jednotlivých činnostiach a ich produktoch priradené iba k metrikám, ktoré sa na danom procese merajú. Druhá úloha tejto entity je v zobrazovaní agregovaných údajov o meraniach pre proces. Ak chceme v aplikácií analyzovať nejaký proces, musíme vybrať metriku, podľa ktorej ho budeme analyzovať. Atribúty *Vrchny_limit* a *Spodny_limit* slúžia ako ukazovatele maximálnej a minimálnej prípustnej hodnoty merania pre daný proces. Pre vybranú dvojicu proces – metrika je možné tieto hranice určiť a potom bude aplikácia sledovať, či sú dodržiavané – užívateľ bude upozornený na procesy, činnosti a ich produkty, kde limity nie sú dodržané. Ak pri danej metrike nemá význam sledovať maximálnu resp. minimálnu prípustnú hodnotu, stačí oba tieto atribúty nastaviť na hodnotu 0.



Obr. 4.13: Entita Proces_Metrika.

Meranie je entita reprezentujúca samotné merania na činnostiach a ich produktoch. Meranie je jednoznačne identifikované atribútom *meranieid*. Dôležité údaje, ktoré musia byť pri meraní zaznamenané sú: nameraná hodnota (*Namerana_hodnota*), nástroj, ktorým bolo meranie vykonané (*Meraci_nastroj*) a dátum merania (*Datum*). Každé meranie musí byť priradené činnosti alebo produktu činnosti (*ArtefaktRef*), metrike (*MetrikaRef*) a užívateľovi, ktorý toto meranie vykonal (*Vykonavatel*). Metrika, ktorej sa meranie týka, musí byť priradená procesu, pod ktorý patrí artefakt, na ktorom bolo meranie vykonané. Zároveň meranie môže vykonať iba užívateľ, ktorý pracuje na projekte, v rámci ktorého sa nachádza artefakt, na ktorom bolo meranie vykonané.

Obr. 4.14: Entita Meranie.

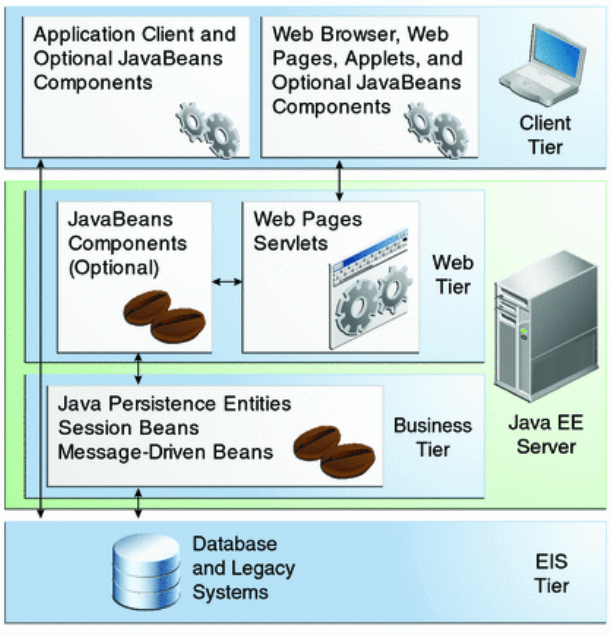
4.4 Implementácia systému pre podporu metrík

Ako implementačná platforma bola vybraná Java EE 6. Hlavné dôvody k výberu tejto platformy boli: vysoká štandardizácia jazyka Java, existencia kvalitného API, viacero integrovaných vývojových prostredí (NetBeans, Eclipse), a hlavne rozšírenosť platformy v praktickej sfére.

Platforma Java EE [8] využíva distribuovaný viacvrstvový model pre aplikácie. Jednotlivé aplikačné vrstvy sú (obr. 4.15):

1. Klientská vrstva
2. Webová vrstva
3. Business vrstva
4. EIS (Enterprise Information system) vrstva

Tento model je možné uvažovať ako trojvrstvový resp. štvorvrstvový model. Trojvrstvové pojatie vyplýva z toho, že webová a business vrstva obe bežia na Jave EE serveri, tým pádom tvoria jednu zloženú vrstvu.



Obr. 4.15: Štvorvrstvový model platformy Java EE [8].

Klientská vrstva môže byť tvorená buďto webovým alebo aplikačným klientom. Pod webovým klientom rozumieme dve zložky: dynamické webové stránky, generované komponentmi vo webovej vrstve a internetový prehliadač, ktorý zobrazuje obsah stránok posielaných zo serveru. Webový klient je možné ho nazvať ľahkým klientom, pretože sám o sebe nerobí žiadne výpočetne náročné úkony (komunikácia s databázou, zložité algoritmy, ...), len posiela požiadavky na server, kde tento výpočet prebehne a zobrazí výsledky. Aplikačný klient je vo forme malej aplikácie, ktorá beží na klientskom počítači. Klient nemusí byť nutne napísaný v jazyku Java a komunikuje priamo s business vrstvou.

Webová vrstva je založená na **servletoch**. Sú to triedy, resp. ich instancie, ktoré spracovávajú požiadavky na server a generujú odpovede. Pre jednoduchší vývoj platforma Java EE poskytuje technológie **JavaServer Pages** (JSP) a **JavaServer Faces** (JSF). JSP poskytuje prirodzenejší spôsob generovania obsahu odpovedí ako samotné servlety. JSF je založená na servletoch a JSP a poskytuje solídny framework pre prácu s webovým obsahom.

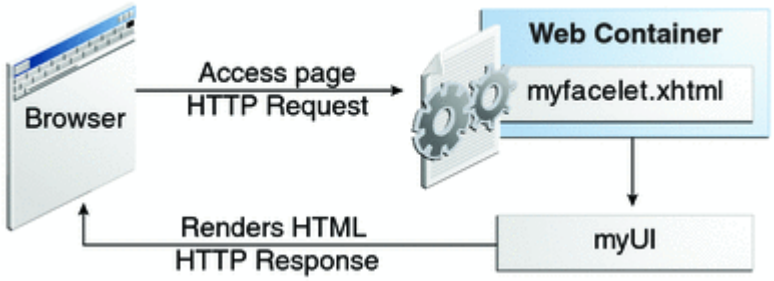
Business vrstva obsahuje logiku aplikácie. Prijíma spracované požiadavky a na ich základe riadi beh aplikácie. Okrem toho vrstva zaobstaráva komunikáciu s databázou – ukladanie dát z klienta a načítavanie dát na odpovede pre klienta. Business vrstva je najčastejšie tvorená pomocou **Entertprise JavaBeans** (EJB).

EIS vrstva najčastejšie obaľuje databázové systémy, ERP systémy, spracovávanie transakcií a podobne.

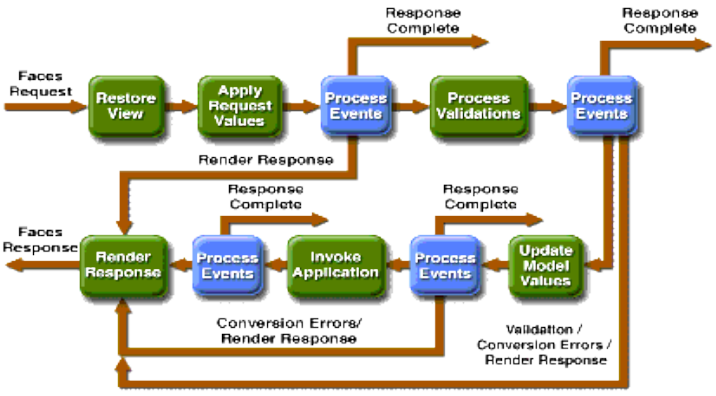
Pri implementácií systému pre podporu metrík bola zvolená web technológia JSF. Technológia JSF poskytuje API pre reprezentáciu komponentov a správu ich stavov, spracovávanie udalostí, validáciu údajov na strane serveru, konverziu dát, prostriedky na definovanie navigácie medzi stránkami, podporu prenositeľnosti, a prostriedky pre pridávanie komponentov do webových stránok a ich napojenie na serverové objekty. Typicky obsahuje aplikácia využívajúca JSF:

1. Webové stránky vytvorené pomocou tagov (či už vlastných alebo podporovaných JSF)
2. Managed beans, ktoré podporujú základné funkcie (spracovanie údajov), zachytávanie udalostí (interceptor) a pod.
3. Súbor *web.xml*, ktorý ukladá nastavenia pre nasadenie.
4. Konfiguračné súbory (formát .xml), ktoré popisujú ďalšie vlastnosti aplikácie (navigácia medzi stránkami, registrácia converterov) ako napr. *faces-config.xml*.
5. Špecifické objekty aplikácie (validátory, konvertory, listeneri).
6. Vlastné tagy pre reprezentáciu špecifických objektov na stránke.

Princíp fungovania JSF [9] je zobrazený na obr. 4.16 a 4.17. Webový prehliadač pošle žiadosť (request) na server. Vtedy sa začína životný cyklus žiadosti (obr. 4.17). Po ukončení životného cyklu je generovaný pohľad (myUI na obrázku), ktorý je už vo forme spracovateľnej prehliadačom a ten je zaslaný ako odpoveď.



Obr. 4.16: Odpoveď na žiadosť na server [8].



Obr. 4.17: Životný cyklus žiadosti JSF [9].

V prvej fáze – obnovenie pohľadu (restore view) – JSF vybuduje nový pohľad stránky. Sú napojené všetky potrebné validátory, komponenty a zachytávače udalostí. Všetky potrebné informácie pre danú žiadosť sú uložené v objekte *FacesContext*, ktorý je prístupný všetkým komponentom, validátorom, konvertorom a zachytávačom udalostí. Ak je to prvá žiadosť na danú stránku vytvorí sa prázdny pohľad a postúpi na generovanie odpovede. Pohľad je doplnený o elementy pri neskorších prístupoch. Ak sa jedná o neskorší prístup na stránku (postback) JSF obnoví pohľad pomocou informácií uložených na klientovi alebo serveri.

V druhej fáze – aplikovanie hodnôt žiadosti (apply request values) – sú údaje v žiadosti skonvertované a na ich základe sú nastavené nové hodnoty komponentov. Potreba konverzie vyplýva z toho, že na stránke sú informácie uložené v textovej podobe. Každý objekt, alebo hodnotu je nutné presne určiť pomocou textového reťazca, aby aplikácia pracovala so správnymi údajmi. Na tento účel slúžia konvertory – objekty implementujúce rozhranie *Converter*. Ako príklad by sme mohli uviesť konvertor pre databázový objekt. Na stránke je reprezentovaný unikátnym reťazcom (napríklad primárny kľúč). Konvertor získa tento identifikátor a na jeho základe vezme z databáze relevantný objekt. Ak pri konverzii nastane chyba, je vytvorená odpoveď, ktorá obsahuje pôvodnú stránku s informáciou o chybe.

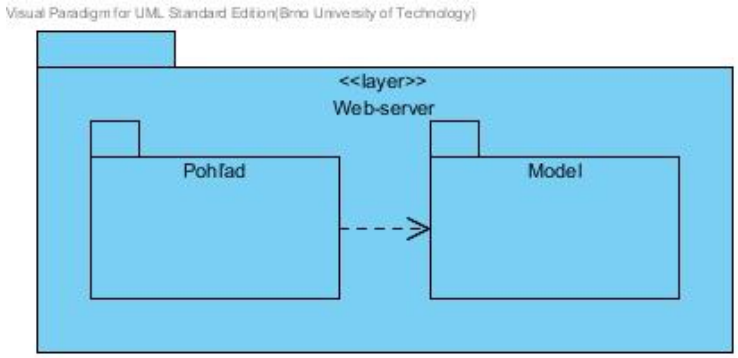
V tretej fáze – validačná fáza (process validation phase) – sa validujú nové hodnoty. Hodnoty sú validované pomocou validátorov – objekty implementujúce rozhranie *Validator*. Existuje veľké množstvo vstavaných validátorov v JSF – napr. validátor dĺžky reťazca textového poľa alebo číselný validátor pre rozsah čísla a ďalšie. Okrem vstavaných môže programátor vytvoriť a použiť aj vlastné. Ak nastanú pri validácii nejaké chyby, informácie od nich sú pridané k informáciám o chybách z predošlej fázy a je generovaná pôvodná stránka so vzniknutými chybami.

V štvrtej fáze – obnovenie hodnôt modelu (update model value phase) – sa podľa nových údajov obnovujú hodnoty v modeli. To znamená, že ak prebehla konverzia a validácia úspešne, tie hodnoty komponentov, ktoré sa viažu na objekty modelu (business vrstvy) sú prenesené na túto vrstvu. Ak sa tu vyskytnú chyby (použitie zlého konvertora, tým pádom priradenie nekompatibilných objektov) generuje sa pôvodná stránka s chybami, podobne ako pri chybách vo validácii.

V piatej fáze – fáza vyvolania aplikácie (invoke application phase) – sú spracované všetky udalosti typu podanie formy, alebo prechod na inú stránku.

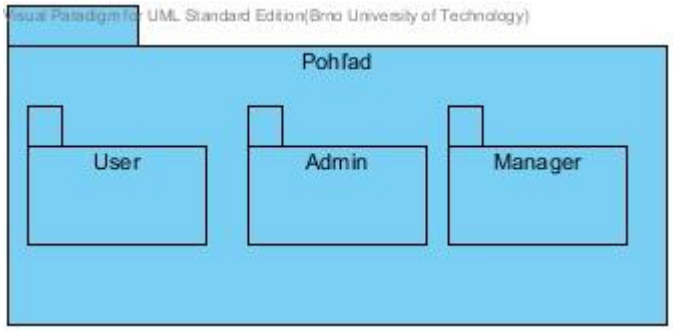
V poslednej fáze – generovanie odpovede (render response phase) – je vytvorená samotná odpoveď na žiadosť na základe použitej technológie (napr. JSP). Ak boli zachytené nejaké chyby počas predošlých fáz, je generovaná pôvodná stránka s informáciami o chybách.

Na základe zvolenej platformy a popísaných technológií bolo nutné zmeniť návrh aplikácie, aby odpovedal použitým prostriedkom. Oproti pôvodnému návrhu (obr. 4.4) sa udiali zmeny vo vrstve Web-server. Nový návrh je zobrazený na obr. 4.18.



Obr. 4.18: Finálny návrh architektúry.

Balíček pohľad obsahuje JSF stránky, z ktorých sú generované odpovede pre klienta. Sú tu obsiahnuté tri balíčky (obr. 4.19): User, Manager a Admin. Jednotlivé balíčky majú chránenú úroveň prístupu. Stránky *index.xhtml, login.xhtml* a *logerror.xhtml* sa nachádzajú v balíčku *Pohľad* a prístup do nich majú všetci užívatelia, aj tí neprihlásení. Po prihlásení je užívateľ na základe svojho umiestnenia v skupine (User, Manager, Admin) presmerovaný do zodpovedajúceho balíčka.

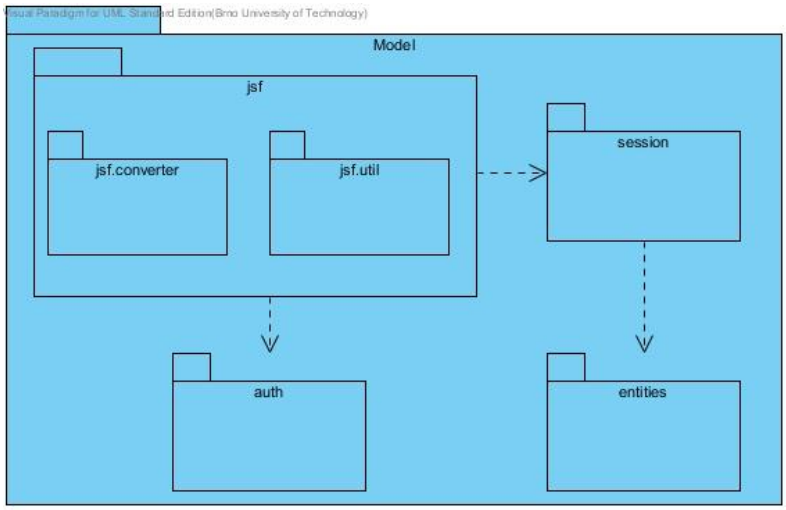


Obr. 4.19: balíček Pohľad.

Balíček model obsahuje logiku aplikácie (obr. 4.20). Je členený do viacero častí. Základnú úroveň tvorí balíček *entities*. Ten obsahuje entitné triedy (značené anotáciou @Entity), ktoré reprezentujú tabuľky v databáze. Tieto triedy sú trvalé a využívajú technológiu Java Persistance API (JPA). Práve vďaka tejto technológii je možné dosiahnuť trvalosť objektu, bez nutnosti vytvorenia EJB modulu.
Následne sa v aplikácii nepracuje priamo s databázou, zadávaním príkazov SQL, ale cez objekt *EntityManager* pomocou jazyka JPA query language. Jednotlivé triedy balíčka sú:

1. Artefakt
2. Groups
3. Meranie
4. Metrika
5. Proces
6. ProcesMetrika
7. ProcesMetrikaPK
8. Projekt
9. ProjektZamestnanec
10. ProjektZamestnanecPK
11. Zamestnanec

Triedy *ProcesMetrikaPK* a *ProjektZamestnanecPK* sú klasické Java objekty (Plain Old Java Object - POJO), ktoré slúžia na reprezentáciu zloženého primárneho kľúča pre triedy *ProcesMetrika* a *ProjektZamestnanec*. Každá z entitných tried má definovanú sadu objektov typu *@NamedQuery*. Tieto objekty sú v podstate príkazy v JPA query language, ku ktorým je možné pristupovať pomocou ich špecifického identifikátora. Výhodou týchto príkazov je, že príkazy týkajúce sa danej entity sú uložené na jednom mieste a v kóde sa potom nenachádzajú duplicity.



Obr. 4.20: Balíček Model.

Balíček *session* obsahuje triedy, ktoré slúžia na manipuláciu s údajmi v databáze pomocou JPA. Balíček obsahuje nasledujúce triedy:

1. AbstractFacade
2. ArtefaktFacade
3. GroupsFacade
4. MeranieFacade
5. MetrikaFacade
6. ProcesFacade
7. ProjektFacade
8. ProjektZamestnanecFacade
9. ZamestnanecFacade

V tomto balíčku bol využitý návrhový vzor fasáda. Všetky triedy obsiahnuté v balíčku rozširujú triedu *AbstractFacade*. Táto trieda implementuje základné operácie, ako pridanie entity, odobranie entity, výber zoznamu entít a podobne. Jednotlivé triedy spravujúce manipuláciu špecifických entít (*ArtefaktFacade* spravuje manipuláciu s entitami typu *Artefakt* atd.) sú rozšírené o pokročilú funkčnosť na základe potreby balíčka *jsf*. Práve tu sú využívané objekty *@NamedQuery* na prístup k údajom v databáze. Okrem toho je tu využitá aj možnosť programového vytvárania JPA query language príkazov. Tento prístup má výhodu v tom, že daný príkaz môžeme meniť na základe daných argumentov metódy. Keby sme nechceli využiť tento prístup, museli by sme pre každú kombináciu argumentov vytvoriť vlastný príkaz. Na druhú stranu vytvárať príkazy programovo je pomerne náročné a pri použití tohto prístupu je možnosť nového použitia minimálna.

Balíček *jsf* obsahuje hlavnú výpočetnú logiku aplikácie. V samotnom balíčku sa nachádzajú triedy, ktoré uchovávajú komponenty balíčku *Pohľad*, spracovávajú požiadavky užívateľa, zabezpečujú navigáciu medzi stránkami a zachytávajú udalosti. Tieto triedy majú formu *@ManagedBeans* a platnosť rozsahu *@SessionScoped*. To znamená, že instancie týchto objektov pretrvávajú počas jedného sedenia (session) s užívateľom. Okrem spracovávania požiadaviek z úrovne pohľadu obsahujú aj modelovú logiku aplikácie. Jednotlivé triedy balíčku *jsf* sú:

1. ArtefaktController
2. GroupsController
3. MeranieController
4. MetrikaController
5. ProcesController
6. ProjektController
7. ProjektZamestnanecController
8. ZamestnanecController

Ako je vidieť, jednotlivé triedy spracovávajú požiadavky podľa entít, ne ktoré sa tieto požiadavky vzťahujú. Tieto triedy využívajú vo veľkej miere balíčku *jsf.util*. Balíček *jsf.util* obsahuje triedy:

1. CSVHelper
2. ChartHelper
3. JsfUtil
4. LinearRegression
5. PaginationHelper
6. passwordHelper

Trieda *CSVHelper* ponúka rozhranie na prácu so súbormi vo formáte *.csv*. Jednou z podmienok pre aplikáciu bola schopnosť vkladania dát a ich výstupu. Grafické rozhranie síce poskytuje možnosť manuálneho vkladania aj výberu dát, ale pre väčšie množstvá dát je spracovávanie po položkách neprijateľné. Preto bola implementovaná možnosť spracovávať súbory s meraniami. Tieto súbory majú presne nadefinovaný formát: musia mať 5 stĺpcov (date, value, metric, measurer, tool), ktoré obsahujú údaje o meraní. Poradie týchto stĺpcov je ľubovoľné. Ako oddeľovač je použitý znak *;* . Každý riadok potom reprezentuje jedno meranie. V rovnakom formáte je poskytovaný aj výstup aplikácie.

Trieda *ChartHelper* poskytuje rozhranie na prácu s grafmi. Ďalšou významnou funkciou programu je schopnosť grafickej reprezentácie nameraných dát. Túto reprezentáciu umožňuje práve trieda *ChartHelper*. Na jej implementáciu bola využitá knižnica JFreeChart. Druhy zobrazovaných grafov a ich význam bude vysvetlený neskôr.

Trieda *JsfUtil* poskytuje rozhranie pre zobrazovanie chybových hlásení. Ak pri spracovaní dát, konverziách alebo vyhodnoteniach, prípadne za behu aplikácie dôjde k nejakej chybe, táto je pomocou prostriedkov triedy *JsfUtil* zobrazená v zrozumiteľnej podobe užívateľovi.

Trida *LinearRegression* poskytuje rozhranie na výpočet lineárnej regresie. Táto je využívaná pri tvorbe grafov. Oddelenie tvorby grafu a výpočtu štatistickej funkcie umožňuje jednoduché pridávanie ďalších štatistických funkcií s väčšou výpovednou hodnotou.

Trieda *PaginationHelper* zabezpečuje stránkovanie údajov. Ako bolo spomenuté, objekty balíka *jsf* pretrvávajú počas celého sedenia s užívateľom. Aby sa zamedzilo nadmernej spotrebe pamäti, bol použitý princíp stránkovania. Ak chce užívateľ zobraziť nejakú skupinu údajov (napríklad merania na procese) zobrazuje a načíta sa vždy iba časť množiny. Tým pádom je v pamäti tiež uložená len časť informácií. Okrem šetrenia pamäti je tento prístup vhodný aj z hľadiska užívateľa. Je podstatne prehľadnejšie mať naraz zobrazených len niekoľko záznamov a potom medzi nimi listovať, než mať na jednej stránke zobrazené všetky údaje, zvlášť ak ich môžu byť stovky.

Trieda *passwordHepler* poskytuje rozhranie na prácu s heslom – hlavne pri zmene hesla (porovnávanie odpovedajúcich si hesiel a pod.).

Posledný balíček *auth* obsahuje iba jednu triedu *AuthLogBean*. Táto trieda zaobstaráva prihlasovanie užívateľa do aplikácie a patričné presmerovanie na základe skupiny.

5 Dosiahnuté výsledky

V kapitole 4.1 boli uvedené 3 hlavné požiadavky na systém pre podporu metrík a niekoľko spresňovacích. Najvhodnejšie vyhodnotenie dosiahnutého výsledku je práve porovnanie s predsavzatiami a požiadavkami.

Prvá požiadavka bola umožniť získavanie metrických dát. V kontexte systému pre podporu metrík to znamená schopnosť vkladať získané dáta o meraniach do systému. Aplikácia túto funkciu podporuje dvoma spôsobmi:

1. Manuálne vkladanie dát „po jednom“.
2. Hromadné vkladanie dát prostredníctvom súborov.

Možnosť vkladania dát do aplikácie je základná a preto ňou disponuje každý užívateľ. Na najnižšej úrovni sú to radoví užívatelia. Oni majú možnosť pristupovať k projektom, na ktorých pracujú a pridávať záznamy o meraniach. Ako bolo spomenuté, vkladanie údajov je umožnené prostredníctvom rozhrania aplikácie (formulár), kde užívateľ musí postupne vyplniť všetky relevantné údaje k meraniu. Druhý spôsob je formou načítania súboru. Ako bolo spomenuté v predošlej kapitole, aplikácia podporuje prácu s údajmi vo formáte *.csv*. Tento formát bol zvolený hlavne kvôli jeho jednoduchosti a širokému nasadeniu. Jednoduchosť formátu prináša hlavne jednoduchú implementáciu a to nielen pre samotný systém pre podporu metrík, ale aj pre iné programy, ktorými by sme dáta chceli ďalej spracovávať. Okrem toho jednoduchosť formátu umožňuje prúdové spracovanie skriptami, pre automatickú manipuláciu. V neposlednej rade bolo myslené aj na manažérov. Súčasť balíku Microsoft Office – Microsoft Excel umožňuje prácu so súbormi vo formáte *.csv*. Tým pádom tvorba prezentácií, dodatočných grafov, alebo pokročilých analýz pomocou tohto softwaru je veľmi jednoduchá.

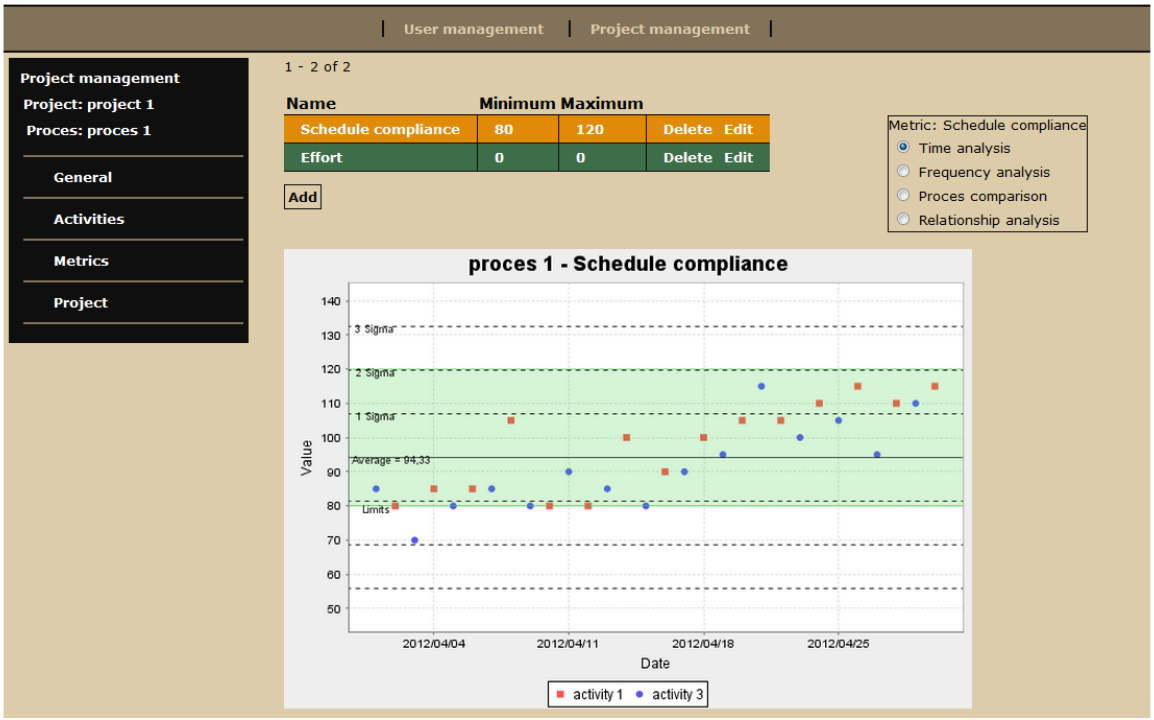
Druhá požiadavka je zameraná na uchovávanie dát. Túto úlohu spĺňa najzákladnejšia vrstva systému – databáza. Samotná aplikácia nepracuje s databázou priamo. Ako je uvedené v predošlej kapitole, využíva prostriedky JPA. Tým pádom je nezávislá na forme samotnej databázy. Pri testovaní bol využitý MySQL server 5.5. Avšak ak pri nasadení aplikácie je k dispozícií iný typ databázy, je postačujúce zmeniť nastavenie aplikácie. V samotnej databáze sú uložené všetky relevantné údaje o užívateľoch, projektoch, metrikách a meraniach, ako popisuje návrh databázy v predošlej kapitole.

Poslednou a najdôležitejšou funkciou je sprostredkovanie dát. Aplikácia umožňuje užívateľovi nielen údaje v surovej, textovej forme, ale aj graficky. Grafické zobrazovanie je umožnené na dvoch úrovniach – na úrovni procesov a na úrovni činností, resp. produktov činností. Dáta sú zobrazované prostredníctvom grafov. Typy grafov a ich významy sú založené na kapitole 3 tohto dokumentu. Systém na úrovni procesov podporuje 4 typy zobrazenia/vyhodnotenia dát:

1. Vyhodnotenie v časovej doméne (Time analysis)
2. Vyhodnotenie vo frekvenčnej doméne (Frequency analysis)
3. Porovnávanie procesov (Proces comparison)
4. Vyhodnotenie vo vzťahovej doméne (Relationship analysis)

Vyhodnotenie v časovej doméne je základný a veľmi dôležitý spôsob vyhodnotenia. Poskytuje nám informácie o priebehu procesu v čase. Samotné vyhodnotenie/zobrazenie dát sa vzťahuje vždy na nejakú zvolenú metriku (postup a GUI je popísané v prílohe). Na obrázku 5.1 je zobrazené samotné vyhodnotenie pre *proces 1* a metriku *Schedule compliance* (dodržiavanie plánu). V grafe sú zobrazené dve série údajov, rozlíšené tvarovo a farebne. Každá séria patrí aktivite, resp. produktu procesu. V ukázanom príklade je daná metrika meraná len na dvoch aktivitách, preto sú zobrazené len dve série. Okrem samotných meraní, je na grafe zobrazených niekoľko ďalších údajov:

1. Priemer (Average)
2. Limity (Limits)
3. Násobky strednej odchýlky (Sigma)



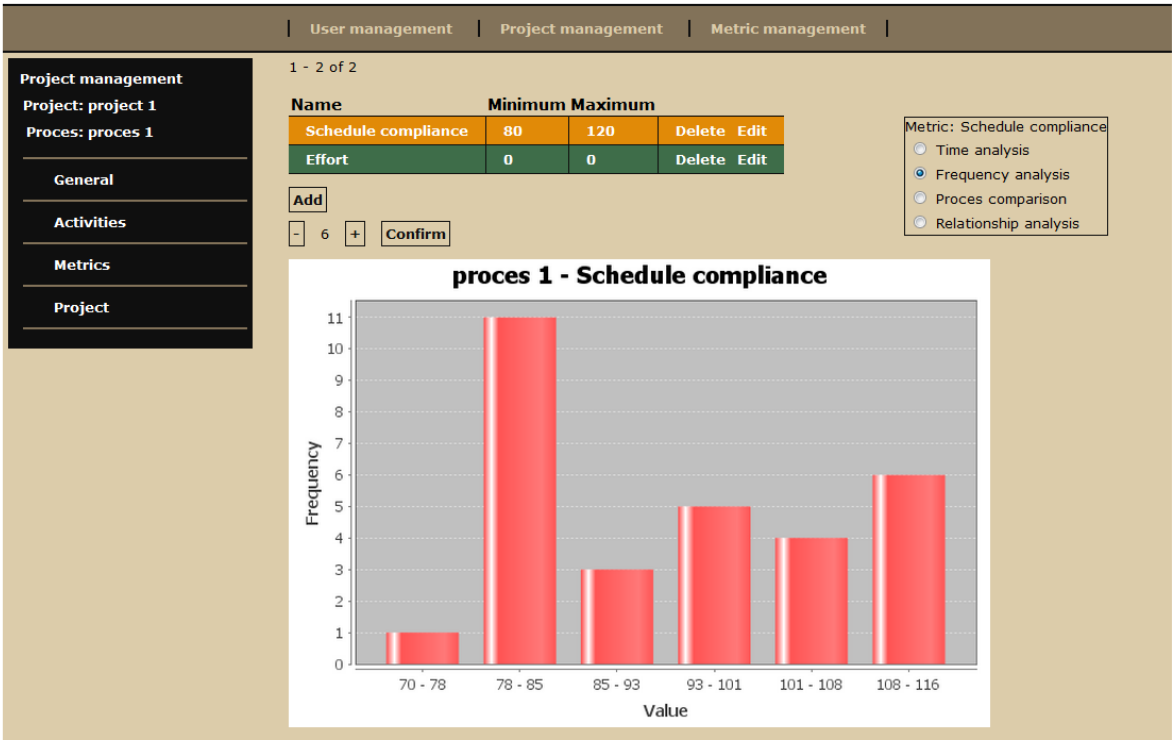
Obr. 5.1: Vyhodnotenie údajov v časovej doméne.

Priemer poskytuje informáciu užívateľovi informácie o strednej hodnote, okolo ktorej sa merania pohybujú. Okrem priemeru bolo možné zvoliť aj iné možnosti reprezentácie strednej hodnoty: modus, medián. Bol však zvolený priemer, pretože vo väčšine prípadov má najväčšiu výpovednú hodnotu a jeho použitie je rozšírené, tým pádom často aj vítané – ľudia radi pracujú s tým, čo už poznajú. Limity sú funkcia aplikácie, ktorá vyšla z jednej z požiadaviek na systém: „mala by byť zahrnutá automatická detekcia problémov“. Užívateľ – konkrétne manažér projektu, alebo administrátor – majú možnosť nastaviť pre zvolenú metriku a zvolený proces maximálnu a minimálnu prípustnú hodnotu (bližšie popísané v prílohe). Ak je zadané nejaké rozmedzie, aplikácia zisťuje existenciu meraní mimo toto rozmedzie a informuje užívateľa. Táto informácia je prístupná taktiež len na úrovniach manažéra a administrátora a funguje takto:

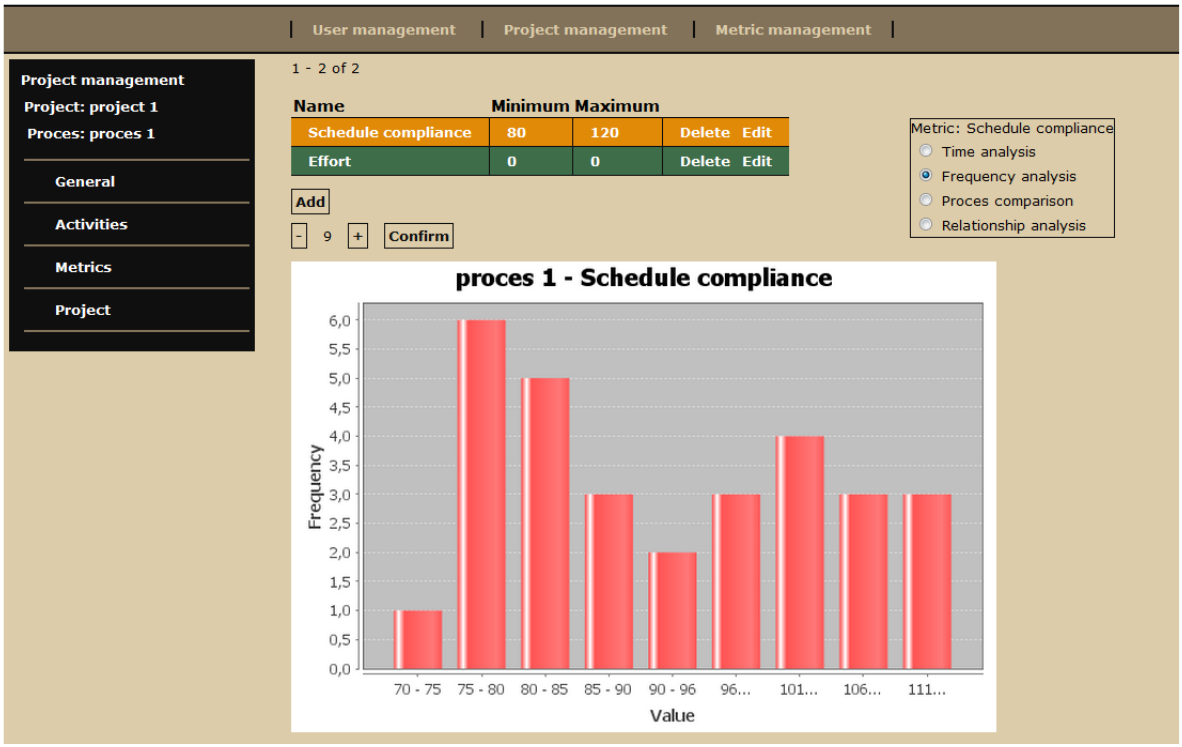
- Ak sú všetky merania v zadanom rozmedzí, je proces/aktivita/produkt/metrika znázornená zelenou farbou.
- Ak je nad 50% meraní v zadanom rozmedzí je proces/aktivita/produkt/metrika znázornená žltou farbou.
- Ak je nad 50% meraní mimo zadaného rozmedzia je proces/aktivita/produkt/metrika znázornená červenou farbou.

Takto má užívateľ okamžitú a prirodzenú informáciu o stavu procesu/aktivity/produktu/metriky. Tieto limity sú v grafe zobrazené zeleným rozmedzím. Posledným dodatočným údajom sú násobky strednej odchýlky. Tieto hodnoty určujú „zdravie“ procesu. Ideálny proces by mal mať merania s normálnym rozložením (kapitola 3). Pre normálne rozloženie poznáme empirické pravidlo troch sigma – 68% hodnôt normálneho rozloženia je v rozmedzí +- σ od priemeru, 95% hodnôt je v rozmedzí 2σ a 99,7% hodnôt je v rozmedzí 3σ. Takto môžeme hneď zbadať, ako je na tom proces ohľadne „zdravia“.

Vyhodnotenie vo frekvenčnej doméne ešte zvýrazňuje princíp popísaný pri strednej odchýlke. Ako bolo povedané, ideálny proces by sa mal približovať normálnemu rozloženiu. Vo frekvenčnej analýze sú zobrazené merania formou histogramu. Toho je možné vyčítať, aké rozloženie proces skutočne má. Príklad histogramu je na obrázku 5.2.



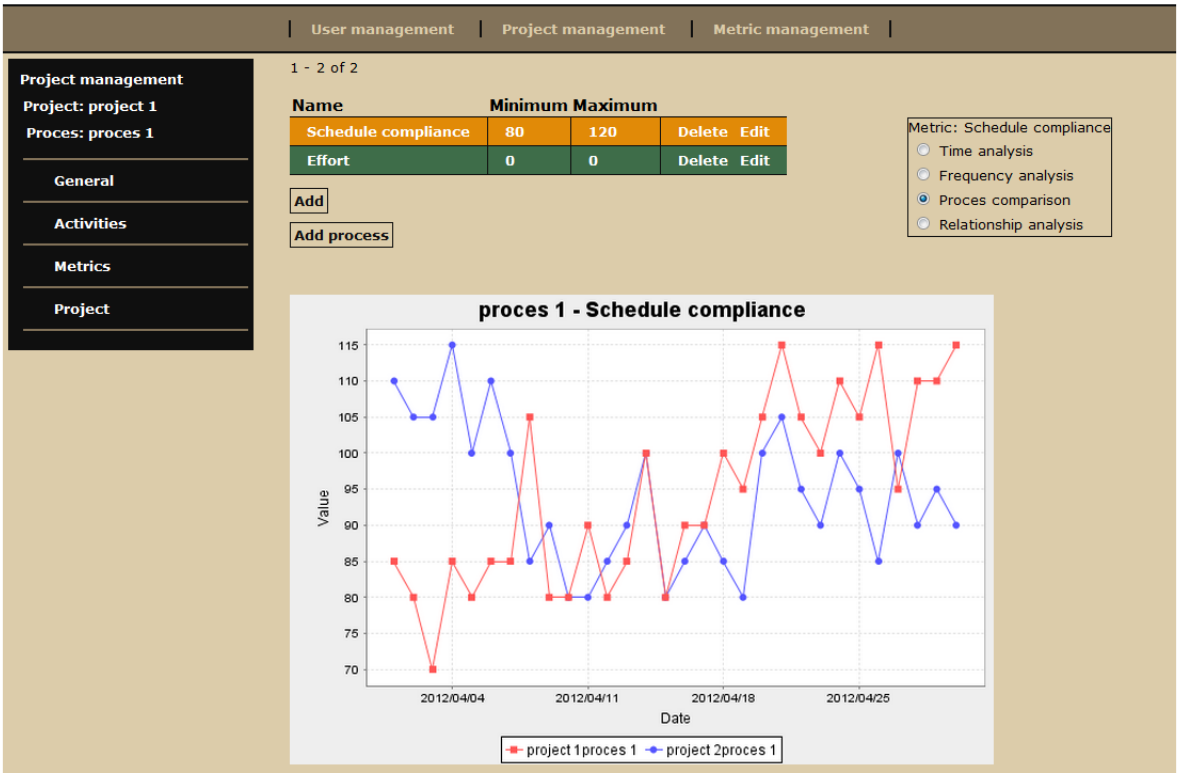
Obr. 5.2: Vyhodnotenie vo frekvenčnej doméne (6 stĺpcov histogramu).



Obr. 5.3: Vyhodnotenie vo frekvenčnej doméne (9 stĺpcov histogramu).

Pri histograme je veľmi dôležitý počet stĺpcov, pretože jeho zmenou sa často zmení aj význam. Aplikácia volí ako základný počet stĺpcov druhú odmocninu z rozmedzia hodnôt. Rozmedzie hodnôt chápeme, ako rozdiel najväčšej a najmenšej hodnoty v grafe. Samozrejme, že tento počet nemusí byť nutne vždy vhodný, existuje množstvo prístupov k výpočtu ideálneho počtu stĺpcov. Preto bolo nutné umožniť užívateľovi počet stĺpcov meniť. Vzhľadom na to, že aplikácia podporuje ukladanie celočíselných hodnôt, maximálny počet stĺpcov je rovný rozmedziu hodnôt. Minimálny počet stĺpcov je 1. Ako príklad zmeny grafu pri zmene počte stĺpcov je uvedený obrázok 5.3 kde je miesto automatickej hodnoty 6 stĺpcov zvolených stĺpcov 9.

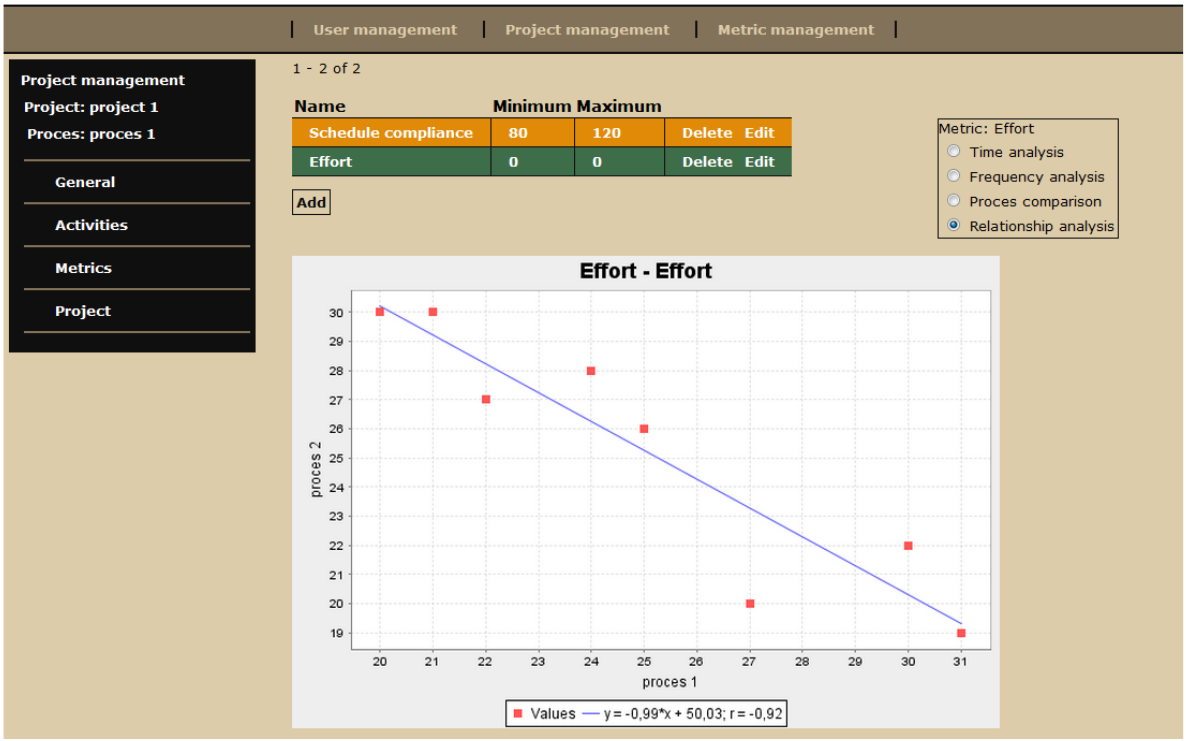
Potreba porovnávania procesov vyplynula z požiadavky: „existencia možnosti zobrazovania informácií pomocou grafov a porovnávanie týchto informácií medzi rôznymi projektmi“. Užívateľ si môže navoliť, ktorý proces chce porovnávať so súčasne zvoleným procesom (postup v prílohe). Vzhľadom na to, že je veľmi pravdepodobné, že procesy prebiehajú v iných časových úsekoch, je porovnávaný proces automaticky prevedený na časovú os analyzovaného procesu. Do samotného grafu (obr. 5.4) je možné pridať ľubovoľné množstvo procesov z rôznych projektov. Samozrejme zvolenie optimálneho množstva vzhľadom na prehľadnosť je na užívateľovi. Ako bolo spomenuté už predtým, ak analyzujeme proces, analyzujeme ho vždy vo vzťahu k nejakej metrike. Preto je možné do grafu pridávať iba tie procesy, na ktorých je vybraná metrika tiež meraná.



Obr. 5.4: Porovnávanie procesov.

Vyhodnotenie vo vzťahovej doméne je posledný možný spôsob analýzy na úrovni procesov a zároveň najzložitejší. Úlohou tohto vyhodnotenia je zistiť, či sa nejaké dve metriky na dvoch procesoch ovplyvňujú, či je medzi nimi vzťah. Táto veta má však veľa možných interpretácií. V prvom rade je nutné si stanoviť rozmedzie, aké procesy môžeme navzájom porovnať, resp. dať do vzťahu. Má význam dávať do vzťahu procesy z dvoch rozličných projektov? Odpoveď na túto otázku nie je celkom jasná. Software projekty sú väčšinou, ako projekty všeobecne, jasne vymedzené a sebestačné. Avšak môže nastať situácia, že jeden projekt je závislý na inom projekte. Príkladom takejto situácie je, že jeden projekt sa týka vývoja informačného systému pre nejakého zákazníka. Úlohou druhého projektu by mohla byť tvorba špecifického software pre toho istého zákazníka, ktorý by nejakým spôsobom interagoval s dodaným informačným systémom. Môžu sa procesy týchto dvoch projektov

navzájom ovplyvňovať? Ja možné, že áno. Ak by sme si napríklad dali do vzťahu úsilie vynaložené na návrh IS a úsilie vynaložené na návrh špecifického software, mohli by sme nájsť závislosť – čím lepšie je navrhnutý IS, tým ľahšie je navrhnúť software, ktorý s ním má spolupracovať. Otázne je, či by závislosti existovali na úrovni metrík vývoja software, alebo na úrovni metrík samotného produktu (napr. kvalita podľa ISO 9000). Takéto prípady sú veľmi špecifické a zriedkavé, preto s nimi aplikácia nepočíta, a vo vzťahovej doméne je možné porovnávať iba procesy z toho istého projektu. Ďalšou otázkou je či má význam dávať do vzťahu odlišné, resp. rovnaké metriky. Tu je odpoveď jasnejšia. Napríklad je možné očakávať, že úsilie spotrebované na návrh jedného komponentu programu ovplyvní úsilie na návrh iného komponentu. Rovnako je pravdepodobné, že dve rôzne metriky sa navzájom ovplyvňujú – napríklad úsilie vynaložené na programovanie a veľkosť kódu, resp. modulu. Preto je nutné umožniť užívateľovi porovnať navzájom dve ľubovoľné metriky. Nakoniec zostáva otázka automatizácie procesu vzťahovej analýzy. Všetky predošlé analýzy boli pomerne ľahko automatizovateľné – užívateľ si zvolil proces, resp. metriku a spracovali sa všetky relevantné údaje. V tomto prípade je nutné určiť, ktoré údaje sú relevantné, a ktoré už nie. Okrem toho ten, kto vyberá údaje na porovnanie musí mať informácie o kontexte údajov. Napríklad chceme na dvoch procesoch *a*, *b* zistiť vzťah nejakých dvoch metrík – veľkosť a úsilie. Do grafu, ktorý nám zobrazuje závislosť sú vykreslované body. Každý bod odpovedá dvojici meraní, jedno meranie z procesu *a*, jedno meranie z procesu *b*. Ako je možné na úrovni aplikácie zistiť, ktoré dve merania majú medzi sebou súvislosť? Ako sa má aplikácia zachovať, keď je medzi meraniami časový nesúlad, alebo posun? Čo sa má vykonať, keď tieto dva procesy nemajú v daných metrikách rovnaký počet meraní? Tieto a ďalšie otázky riešiť na úrovni aplikácie, definovaným algoritmom je zložité ale hlavne nepresné. Údaje poskytované analýzou by mali byť presné a relevantné, pretože slúžia na podporu rozhodnutí pre manažéra! Preto nie je prípustné aby dochádzalo k závažným chybám, z ktorých by boli vyvodené chybné úsudky, vedúce až k zlyhaniu projektu.



Obr. 5.5: Vyhodnotenie vo vzťahovej doméne.

Z týchto dôvodov bol zvolený prístup, kde užívateľ musí kontext vložiť ručne. To znamená že užívateľ si vlastnoručne vyberie dvojice meraní, ktoré majú reprezentovať bod v grafe. Tento prístup kladie dodatočné nároky na užívateľa – hlavne časové, pretože postup je prácnejší – avšak čiastočne eliminuje možnosť vzniku chybného výstupu. Čiastočne preto, že užívateľ tvorí graf sám. To znamená že k jeho vytvoreniu potrebuje pomerne veľkú mieru voľnosti (podrobný proces tvorby je

v prílohe) a tam vzniká priestor pre chyby. Na druhú stranu, ak by aplikácia viac obmedzila schopnosť užívateľa tvoriť graf, tak by boli eliminované niektoré možnosti porovnávania. Bol zvolený kompromis vo voľnosti tvorby grafu – užívateľ si pevne navolí proces a metriku, ktorú chce dávať do vzťahu a potom ľubovoľne vyberá merania zo zvoleného procesu.

Výsledný graf je zobrazený na obrázku 5.5. Jednotlivé body reprezentujú dvojice meraní, ktoré sa dávajú do vzťahu. Modrou čiarou je znázornená lineárna regresia, ktorej rovnica je uvedená v legende grafu. Lineárna regresia bola zvolená ako štatistický nástroj vyjadrenia vzťahu, ak nejaký existuje. Na uvedenom príklade vidíme, že vzťah medzi procesmi existuje a je klesajúci. Do vzťahu boli vzaté dva procesy z rovnakého projektu a u oboch metrika vynaloženého úsilia. Z grafu je vidieť, že čím viac úsilia bolo vynaložené na *proces 1*, tým menej úsilia bolo potrebné vynaložiť na *proces 2*. Lineárna regresia nám hovorí, ako prudko táto závislosť klesá. V tomto konkrétnom prípade sa jedná o takmer priamo úmerné klesanie – o koľko zvýšime úsilie na *procese 1,* o toľko sa nám zníži potrebné úsilie na *procese 2*. A máme informácie o cene – napríklad úsilie vynaložené na *proces 1* stojí menej ako úsilie na *proces 2* – tak vieme urobiť rozhodnutie, že na *proces 1* vynaložíme viac úsilia a tým pádom ušetríme. Okrem samotnej regresie je v legende uvedený aj koeficient *r* spomínaný v kapitole 3. Ten nám udáva hodnotu, nakoľko sú porovnávané veličiny (procesy) vo vzťahu. Jeho hodnota sa pohybuje v rozmedzí <-1,1>. V našom prípade má hodnotu -0,92 a to znamená, že je tu silná negatívna závislosť. Hodnoty koeficientu okolo nuly znamenajú malú až žiadnu viditeľnú závislosť.

5.1 Metriky

Nájsť rozumnú, univerzálnu sadu metrík pre akýkoľvek projekt, alebo akúkoľvek organizáciu je takmer nemožné. Aby bolo meranie efektívne a výsledky z neho pre organizáciu prínosné, mala by každá organizácia vymyslieť vlastnú sadu metrík – či už úplne nových a osobitých, alebo už existujúcich a prevzatých – a aplikovať ich. Systém pre podporu metrík by bol ale nekompletný, keby užívateľom neponúkal žiadne možnosti, nápady, aké metriky použiť. Táto kapitola sa venuje možným metrikám, ktoré už boli niekedy použité, resp. ktoré existujú a je ich možné použiť.

Pri vývoji software sú dôležité produkty z každej fázy: modely z návrhovej fázy, napísané programy z implementačnej fázy, testy z testovacej fázy, dokumenty týkajúce sa dokumentácie a podobne. Existuje mnoho metrík, ktoré sa zaoberajú meraním kódu programu. Jedna zo základných je **veľkosť** programu a s ňou spojené metriky. Medzi dva najpoužívanejšie prístupy merania veľkosti patria:

1. Počet riadkov (Lines of code – LoC)
2. Funkčné body (Function points – FP)

Pri meraní **počtu riadkov** je treba dbať na viacero faktorov [10]. Najdôležitejšie je určiť, ktoré riadky sú relevantné a ktoré nie. V príklade 5.1 máme uvedenú ukážku programu, ktorý má spolu 9 riadkov. To znamená, že by mal mať hodnotu LoC = 9.

```
1     int I = 0;
2     int a[] = new int[10];
3
4     While ( I < 10 )
5     {
6           I = I + 1;
7           // computes value for array a element with index I
8           someFunc(a[I]);
9     }
```

Ak sa na program pozrieme bližšie, zistíme, že počet riadkov, ktoré obsahujú nejaký príkaz – v ktorých sa niečo deje – je len 5 (riadky číslo 1, 2, 4, 6, 8). Riadky 3, 5 a 9 neobsahujú žiadne príkazy, iba znaky (ohraničenie zloženého príkazu) alebo len znak nového riadku. Otázne je, či je

nutné rátať pri počte riadkov aj s takýmito neproduktívnymi riadkami. Jedným možným vylepšením je SLoC – Statement Lines of Code – tu sa počítajú iba tie riadky, ktoré obsahujú nejaký príkaz. Druhý faktor, ktorý je treba zvážiť sú komentáre. Komentáre ako také nemajú vplyv na funkčnosť programu. Na druhú stranu ich napísanie si vyžadovalo isté úsilie a majú vplyv na možné budúce zmeny v kóde (ušetria čas, keď sa programátor chce vracať ku kódu, či už kvôli zmenám alebo kvôli reengineeringu). Potom by sme mohli vyjadriť LoC = NCLoC + CLoC, teda počet riadkov ako súčet počtu nekomentovaných riadkov (NCLoC) a komentovaných riadkov (CLoC).

Samotná metrika LoC nám nesie len informáciu o veľkosti, resp. komplexnosti kódu. Ak z nej chceme vyťažiť čo najviac je vhodné ju kombinovať s inými meranými veličinami:

- Produktivita: LoC/Človekodeň
- Kvalita: Chyby/LoC
- Hustota dokumentácie: Stránky dokumentácie/LoC

Pre lepšie počítanie je vhodné pri väčších zdrojových súboroch použiť miesto LoC metriku KLoC (Kilo Lines of Code), ktorá počíta miesto jednotlivých riadkov stovky riadkov. Medzi hlavné nevýhody tohto prístupu partia veľká závislosť na programovacom jazyku – tým pádom nie je dosť dobre možné porovnávať dva programy napísane v iných programovacích jazykoch) a zložitosť odhadov. Možnosť správne odhadnúť veľkosť programu v LoC pred fázou návrhu, alebo často aj po nej, je veľmi nízka.

Funkčné body [3,4] merajú program z hľadiska funkčnosti. Táto funkčnosť sa berie z hľadiska užívateľa a nie z hľadiska vývojára. Hlavná prednosť funkčných bodov je v tom, že nie sú závislé na jazyku, v ktorom je program implementovaný. To je aj jeden z dôvodov, prečo sú FP používané aj v štandarde ISO.

Pre spočítanie FP je nutné najprv určiť hodnotu neupravených funkčných bodov (Unadjusted Function Points – UFP). Na ich výpočet spočítame tieto elementy:

- Vstupy (mená súborov, položky výberového menu, dáta z iných aplikácií, ...)
- Výstupy (výstup na obrazovku, správy, dáta smerujúce do iných aplikácií, ...)
- Dotazy (vstupy do systému, ktoré sa neukladajú ,ale generujú odpoveď systému)
- Externé súbory (automaticky spracovateľné rozhrania na iné systémy)
- Vnútorné súbory (vnútorné entity systému, ktoré sa vytvárajú, upravujú a rušia v rámci systému)

Po zistení počtu jednotlivých elementov, je každému elementu priradená subjektívna hodnota komplexnosti – jednoduché, stredné, zložité. Potom počty jednotlivých elementov danej zložitosti vynásobíme hodnotou komplexnosti na základe tabuľky 5.1.

Element	**Jednoduchý**	**Stredný**	**Zložitý**
Vstupy	3	4	6
Výstupy	4	5	7
Dotazy	3	4	6
Externé súbory	7	10	15
Interné súbory	5	7	10

Tab. 5.1: Tabuľka ohodnotenia komplexnosti pre UFP [4].

Súčtom vypočítaných súčinov dostávame hodnotu UFP. Aby sme získali hodnotu FP, musíme vynásobiť hodnotu UFP technickým faktorom komplexnosti (Technical Complexity Factor – TCF). Tento je vypočítaný na základe 14 faktorov [4], ktoré sú ohodnotené na stupnici od 0 do 5 na základe ich vplyvu. Potom je spočítaný ich súčet S a hodnota $TCF = 0{,}65 * 0{,}01 * S$.

1. Spoľahlivá záloha a zotavenie
2. Distribuované funkcie
3. Veľké používanie konfigurácie
4. Jednoduchosť používania
5. Komplexnosť rozhrania
6. Znovu použiteľnosť
7. Nasadenie na viacerých miestach
8. Výkon
9. Presuny dát
10. On-line vstup dát
11. On-line update
12. Zložitosť spracovania
13. Jednoduchosť inštalácie
14. Zapracovávanie zmien

Ďalšou formou metrík programu sú **Halsteadove metriky** [3]. Podľa Halsteada je implementácia algoritmu súbor operátorov a operandov, presnejšie povedané ich sekvencia. Halsteadove metriky sú tým pádom založené na 6 merateľných atribútoch:

1. n1 – počet rozličných operátorov
2. n2 – počet rozličných operandov
3. N1 – počet všetkých výskytov operátorov
4. N2 – počet všetkých výskytov operandov
5. n1* – minimálny počet operátorov modulu/programu
6. n2* – minimálny počet operandov modulu/programu

Z nich sú vypočítané nasledujúce metriky [3]:

- Dĺžka programu: $N = N1 + N2$
- Slovná zásoba programu: $n = n1 + n2$
- Objem programu: $V = N * \log_2 n$
- Minimálny objem: $V^* = (2 + n2^*) * \log_2(2 + n2^*)$
- Úroveň programu: $L = \frac{V^*}{V}$
- Náročnosť programu: $D = \frac{1}{L}$
- Odhad úrovne programu: $\hat{L} = \frac{2}{n1} * \frac{n2}{N2}$
- Inteligentný obsah programu: $I = \hat{L} * V$
- Úsilie na programovanie: $E = \frac{V}{L}$
- Čas potrebný na programovanie: $T = \frac{E}{S}$

Pričom v poslednom vzorci S znamená počet jednoduchých diskriminačných operácií vykonaných ľudským mozgom za sekundu (Stroudove číslo).

Metrika výrazne súvisiaca s metrikami, ktoré merajú veľkosť je hustota chýb a atribút, ktorý meriame – počet chýb [4]. Keď hovoríme o chybách je nutné rozlišovať medzi chybami, ktoré sa vyskytujú v programe alebo návrhu (faults) a chybami samotnej aplikácie (zlyhania – failure). Veľké množstvo chýb v programe sa odhalí a odstráni už počas vývoja. Napríklad syntaktické chyby sa odhalia prekladom, sémantické chyby pri skúmaní kódu alebo pri testoch. Na základe týchto chýb tvoríme metriku hustota chýb (vzorec 5.1). Ako vidíme, metrika je silne závislá na metóde, akou meriame veľkosť.

$$Fault\ density = \frac{number\ of\ faults}{size} \qquad \text{(vzorec 5.1 [4])}$$

Ďalší faktor ovplyvňujúci túto metriku sú samotné chyby. Musíme si položiť otázku, aké chyby meriame? Či meriame chyby odhalené počas návrhu, chyby odhalené po dokončení vývoja, opravené chyby a podobne. Okrem samotných chýb nám na metriku vplýva aj postup akým informácie o existujúcich chybách získavame. Je veľmi nepravdepodobné, že počas inšpekcie kódu, alebo testovania sa odhalia všetky chyby v projekte. Preto nám táto metrika môže ponúknuť cenný náhľad na efektívnosť metód hľadania chýb.

Chyby, ktoré pretrvali počas vývoja software sa potom môžu prejaviť ako zlyhania aplikácie. Tieto zlyhania potom vplývajú na **spoľahlivosť** aplikácie. Metrika, ktorá sa často využíva na vyjadrenie spoľahlivosti je priemerná doba medzi zlyhaniami aplikácie (Mean time to failure – MTTF), ktorú je možné spočítať na základe funkcie hustoty pravdepodobnosti zlyhania – *f(t)* (vzorec 5.2).

$$MTTF = \int tf(t)dt \qquad \text{(vzorec 5.2 [4])}$$

Následne je možné vypočítať ďalšie metriky súvisiace so spoľahlivosťou [4]:

- Priemerná doba medzi chybami: $MTBF = MTTF + MTTR$
- Dostupnosť: $Availability = \frac{MTTF}{MTBF} * 100\%$

Kde MTTR je priemerná doba do opravy aplikácie. Vzhľadom na to, že systém pre podporu metrík pri vývoji software sa zameriava hlavne na vývoj a nie na hotový produkt, metriky spoľahlivosti, rovnako ako metriky kvality podľa ISO 9000 nie sú v aplikácií zahrnuté.

Poslednou metrikou, ktorá tu bude spomenutá v súvislosti s veľkosťou kódu je **produktivita** [4]. Pri produktivite je dôležité si uvedomiť, čoho produktivitu chceme merať. Existuje viacero možností: produktivita vývojára, procesu, nástrojov a podobne. Vo všeobecnosti by sme produktivitu mohli vyjadriť, ako podiel výsledku a zdrojov. Napríklad pri produktivite vývojára:

$$Productivity = \frac{size}{effort} \qquad [4]$$

Opäť nastáva situácia, kde je produktivita závislá na metóde merania veľkosti. Aj keď meranie veľkosti formou LoC je zaužívané a konkrétne pri produktivite ponúka „pekné“ čísla, je to veľmi neobjektívny spôsob merania (kvôli nevýhodám spomenutým pri metóde merania veľkosti pomocou LoC). Druhým faktorom, ktorý ovplyvňuje produktivitu je úsilie (effort). V súčasnosti sa pri projektovom riadení kladie veľký dôraz na riadenie a zlepšovanie. Preto meranie úsilia býva často veľmi presné.

Okrem kódu aplikácie, ktorý je výsledkom implementácie, sú súčasťou vývoja softvéru aj modely návrhu a špecifikácie, dokonca často aj vo väčšej miere. Preto treba zvažovať aj použitie metrík, ktoré sa vzťahujú na tieto modely.

Najčastejšie používané modely sú vytvárané v jazyku UML. Modely vytvorené v jazyku UML sú vizuálne reprezentované pomocou diagramov. Jazyk UML obsahuje prostriedky pre popis statickej, aj dynamickej štruktúry. UML podporuje diagramy:

- Diagram tried
- Diagram objektov
- Diagram balíčkov
- Diagram komponentov
- Diagram zloženej štruktúry
- Diagram nasadenia
- Diagram prípadov použitia (Use Case)
- Diagramy interakcie (sekvencie, komunikácie, časovania a prehľadový diagram interakcie)
- Diagram stavového automatu

- Diagram aktivity

Diagramy prípadov použitia sú základnými diagramami. Ich použitie je značné hlavne v počiatočných fázach projektu (špecifikácia a návrh). Pre diagramy prípadov použitia existuje veľké množstvo metrík, ktoré diagramy analyzujú z rôznych hľadísk.

Jednou z možností použitia metrík pre Use case diagramy je odhad komplexnosti systému. Podľa Mearchesi a spol. [11] sú: počet prípadov použitia (Ncu), počet aktérov (Na) a počet *include* a *extend* vzťahov. Na ich základe je navrhnutá zložitosť systému ako:

$$UC4 = K_1 * UC1^2 + UC3 + K_2 * [smm([C]) - smm([E])] \quad [11]$$

Kde sú K1 a K2 konštanty, ktoré musia byť empiricky vypočítané, UC1 počet prípadov použitia v špecifikácií, UC3 je počet komunikácií medzi prípadmi použitia a aktérmi bez redundancií (spôsobené *include* a *extend*). [C] a [E] sú matice s rozmermi $Na * Ncu$, kde element c_{ik} má hodnotu 1 ak aktér *i* má vzťah s prípadom použitia *k*. Rozdiel medzi týmito dvoma maticami je ten, že matica [E] je bez redundancií (spôsobené *include* a *extend*). Operátor *smm* určuje sumu všetkých prvkov matice.

Inou možnosťou použitia metrík pre Use case diagramy je určenie potrebného úsilia pre vytvorenie aplikácie. Jenou z možností, ako navrhol Karner [11] je odhad úsilia pomocou Use case points (UCP), ktoré sú analogické k FP. Výpočet definoval ako:

$$UCP = UUCP * TCF * EF \qquad [11]$$

Kde *UUCP* je počet neupravených UCP a počíta sa ako vážená suma aktérov a prípadov použitia, pričom každý prvok môže mať rôznu komplexnosť. Tým pádom je nutné na základe faktorov (počet krokov v prípade použitia, počet analytických tried vyplývajúcich z prípadu použitia) nutné priradiť jednotlivým elementom váhu. TCF je faktor technickej komplexnosti, ktorý sa počíta na základe 13 faktorov (viď [11]). EF reprezentuje úroveň skúseností projektového tímu.

Druhým najpoužívanejším UML diagramom je diagram tried. V [11] je navrhnutý súbor metrík, ktorých úlohou je analyzovať UML diagram tried za účelom merania štruktúrnej zložitosti. Prednesené metriky sú:

- Počet asociácií: NAssoc
- Počet agregácií: NAgg
- Počet závislostí: NDep
- Počet generalizácií: NGen
- Počet agregačných hierarchií: NAggH (agregácie viacerých úrovní)
- Počet generalizačných hierarchií: NGenH
- Maximálna hĺbka generalizačného stromu: MaxDIT
- Maximálna hĺbka agregačného stromu: MaxHAgg

S predloženými metrikami boli vykonané tri pokusy (v kombinácií s metrikami: počet tried, počet atribútov a počet metód), ktoré sa zamerali hlavne na určenie zrozumiteľnosti, zložitosti úprav a analyzovateľnosti (viac v [11]).

Doteraz spomínané metriky sa týkajú hlavne produktov aktivít. Ak chceme merať aktivity, je nutné sa zamerať na atribúty aktivít. Jedny z najčastejšie meraných atribútov aktivít, z hľadiska projektového riadenia sú:

1. Úsilie
2. Čas
3. Cena

Úsilie vyjadruje potrebnú pracovnú silu, ktorá na projekte pracuje daný čas. Ako jednotky úsilia sú používané človeko-hodiny, človeko-dni alebo človeko-mesiace, podľa rozsahu. Význame merania vynaloženého spočíva okrem vedenia záznamov o úsilí aj v použití pre výpočet iných metrík. Príkladmi takýchto metrík sú:

- Produktivita: $Productivity = \frac{size}{effort}$
- Dodržiavanie plánovaného úsilia : $Effort\ variance = \frac{actual\ effort}{planned\ effort}$

Čas a cena sú, podobne ako úsilie, kritickým ukazovateľom projektov. V softwarových projektoch sa často stáva, že je prekročený ako čas, tak aj rozpočet. Preto je potrebné oba tieto atribúty pozorne sledovať. Metriky, v ktorých je možné využiť tieto ukazovatele sú napríklad:

- Dodržiavanie rozvrhu: $Schedule\ variance = \frac{time\ spent}{time\ planned}$
- Dodržiavanie rozpočtu: $Budget\ variance = \frac{actual\ cost}{planned\ cost}$

Konkrétne systém metrík je nakoniec vždy špecifický podľa potrieb spoločnosti. Ako príklady sú ďalej uvedené niektoré konkrétne plány:

Siemens – čas, veľkosť, kvalita, cena [12]:

- Veľkosť (priamy prístup neurčený)
- Dodržiavanie plánu/rozpočtu
- Cena opravy chýb
- Zmeny vlastností
- Profil zisťovania chýb

Shell [12]:

- Intenzita dodávania projektov
- Rýchlosť dodania
- Hustota chýb
- Časový rozvrh
- Cena za úsilie

Návrh univerzity južnej Kalifornie [13]:

- Postup (ukončené aktivity / plánované aktivity)
- Úsilie (vynaložené úsilie / plánované úsilie)
- Cena (vynaložené prostriedky a odchýlky od plánu)
- Výsledky inšpekcie (počítajú sa *Action items* – chyby zistené inšpekciou a pripomienky zákazníka)
- Hlásenia o problémoch (počet nahlásených problémov)
- Stabilita požiadaviek
- Stabilita veľkosti (dosiahnutá veľkosť / plánovaná veľkosť)
- Využitie počítačových prostriedkov

6 Záver

Keď chceme používať metriky, musíme pamätať na štyri body: návrh systému metrík, získanie údajov, vyhodnotenie údajov a použitie údajov. Návrh systému je špecifický pre danú organizáciu. V najväčšej miere závisí na cieľoch, ktoré si organizácia určí. Podľa toho sa musí zvoliť množina metrík, ktorú chceme pre systém použiť. Možné kombinácie boli predložené v kapitole 5. Keď už sú metriky vybrané je nutné realizovať meranie. Ak boli vybrané nejaké špecifické metriky, ktoré ešte nie sú popísané, treba ich navrhnúť a potom aplikovať. Detaily ohľadne návrhu a aplikácií metrík boli spomenuté v kapitole 2. Po získaní údajov nasleduje ich vyhodnotenie. Formy vyhodnotenia a vizualizácie metrických údajov boli popísané v kapitole 3.

Úlohou diplomovej práce bolo navrhnúť a implementovať systém pre podporu metrík pri projektoch vývoja softwaru. Úlohu sa podarilo splniť a výsledkom je systém MetricsVisage. Tento systém podporuje: import, uchovávanie a reprezentáciu údajov. Aplikácia poskytuje pre reprezentáciu projektov procesný model, kde je projekt reprezentovaný ako súbor procesov, paralelne alebo sériovo bežiacich, ktoré sú zložené z aktivít a ich produktov. Analýza dát je podporovaná prostredníctvom vizualizácie údajov v grafoch. Metrické dáta sú analyzované v troch doménach: časová, frekvenčná a vzťahová doména a na troch úrovniach: úroveň procesov, aktivít a ich produktov. Systém bol vyvinutý v čo možno najvšeobecnejšej miere, aby mohol byť reálne aplikovaný v situáciách, kde organizácie chcú zavádzať systém metrík a hľadajú k tomu vhodné prostriedky.

Vzhľadom na to, že systém má byť pomerne všeobecný je tu veľká možnosť ho v budúcnosti rozširovať a pridávať funkcionalitu. Vysoký prínos by mohlo aplikácií priniesť implementovanie časového riadenia projektov. V súčasnosti existuje veľké množstvo nástrojov na riadenie projektov, v ktorých je časová doméne základ (napr. Microsoft Project). Pridanie časovej domény do systému MetricsVisage by prinieslo viacero výhod. Za prvé by bola umožnená prísna kontrola dátumov meraní. Ak by existoval plán meraní, jeho dodržiavanie, resp. nedodržiavanie by mohlo byť kontrolované automaticky aplikáciou. Potom by užívateľ mohol byť o problémoch informovaný hneď pri prihlásení do aplikácie (hlavne užívateľ skupiny manager). Okrem kontroly vykonávania meraní a možnosti časovej kontroly projektu, by implementácia časovej domény mohla viesť k implementácií automatického zaznamenávania metrík. Už skôr v práci bola popisovaná možnosť automatického počítania nepriamych metrík. Táto možnosť bola v súčasnom kontexte označená za veľmi náročnú až nemožnú. Po zavedení časovej domény, by bolo podstatne jednoduchšie automaticky počítať niektoré metriky, ktoré záležia na časovej doméne. Ideálnym príkladom by bola metrika dodržiavanie plánu. Každý proces, alebo aktivita by mohla mať svoj vymedzený časový rámec podľa plánu a reálny časový úsek priebehu (napr. pomocou dátumu začiatku a konca). Rozdiel týchto časových údajov by potom reprezentoval mieru dodržiavania plánu pre daný proces, resp. aktivitu. Ak by bolo potrebné podrobnejšie zaznamenávať údaje o dodržiavaní plánu, ideálnym riešením by bola implementácia míľnikov (milestone).

Ako bolo naznačené pri rozšírení o časovú doménu, všetky rozšírenia by mali smerovať k maximálnej automatizácií systému. Jedným z možných vylepšení v tomto ohľade, je pridanie nového vzťahu medzi aktivitami a ich produktmi. V súčasnom stave systému sú produkty vnímané, ako výstupy aktivít a zväčša sa na ne vzťahujú aj iné metriky. Možné vylepšenie by bolo, keby boli produkty aktivít vnímané aj ako vstupy pre iné aktivity. Tento vzťah sa v reálnych projektoch vyskytuje často – dokument špecifikácie je vstupom pre návrh, model návrhu je vstupom pre implementáciu a zdrojový kód z implementácie je zas vstupom pre testovanie. Takýto typ vzťahov medzi entitami v rámci systému by automatizácií systému pomohol hlavne z hľadiska vyhodnotenia. Ak by sme mali spojenie (kontext) medzi dvoma rôznymi produktmi aktivít boli by sme schopní automatizovať vzťahové vyhodnotenie metrických údajov medzi týmito dvoma entitami.

Ako bolo uvedené skôr, na produkty a aktivity sa často vzťahujú iné metriky. Napríklad metrika veľkosť sa na aktivitu, čo je nejaká činnosť aplikuje veľmi ťažko, ak je to vôbec možné. Naopak na

produkt tejto aktivity je aplikovaná veľmi ľahko – napr. ako počet riadkov. Preto by sa mala zvažovať aj možnosť klasifikácie metrík. Jednotlivé metriky by mohli byť rozdelené do rôznych kategórií, podľa predmetu merania, alebo aplikácie. Prirodzene rôzne metriky by potom potrebovali pravdepodobne aj odlišné atribúty. Príkladom môžu byť kumulatívne metriky. Tieto metriky sú vypočítané tak, že pri jednotlivých meraniach sa hodnoty meraní postupne pričítajú – kumulujú. Príkladom takejto metriky by mohlo byť úplné vynaložené úsilie (Total effort spent). Samozrejme pri metrikách tohto typu je možné výraznou mierou automatizovať – pri pridaní merania sa aktualizuje len nová hodnota.

Ak by sme chceli implementovať perfektný systém pre podporu metrík, bolo by na to treba veľa času, ľudí a hlavne skúseností. Takisto perfektný systém pre jednu organizáciu môže byť nepoužiteľný pre inú organizáciu, alebo inú metodológiu vývoja. Preto bola úloha vytvoriť systém so širokým použitím veľmi náročná, ale aj výrazne poučná.

Literatúra

[1] Standish group. *New StandishGroup report shows more projects failing and less successful projects*. 23. apríla 2009. [cit. 18. novembra 2011]. Dostupné na URL: < http://www1.standishgroup.com/newsroom/chaos_2009.php>.

[2] Ebert, Ch. et al. *Best practices in software measurement*. Springer Berlin Heidelberg, Nemecko, 2005. ISBN 3-540-20867-4.

[3] Abran, A. *Software metrics and software metrology*. John Wiley & Sons, Inc. New Jersey, 2010. ISBN 978-0-470-59720-0.

[4] Fenton, N. E. Pfleeger, S. L. *Software metrics: a rigorous and practical approach*. Druhé vydanie. PWS publishing company, Boston, 1997. ISBN 0-534-95425-1.

[5] Pandian, C.R. *Software metrics: a guide to planning, analysis and application*. Auerbach publications, 2004. ISBN 0-203-49607-8

[6] Wiens, E. G. *Egwald statistics: probability and stochastic processes.* 1. decembra 2001. [cit. 26. novembra 2011]. Dostupné na URL: <http://www.egwald.ca/statistics/probability.php>.

[7] Bourke, P. *Miscellaneous functions.* Január 2001. [cit. 26. novembra 2011]. Dostupné na URL: <http://paulbourke.net/miscellaneous/functions/>.

[8] Jendrock, E. et al. *The Java EE 6 Tutorial*. Oracle, 500 Oracle Parkway , Redwood City, U.S.A. 2012. Apríl 2012. [cit. 10 apríla 2012]. Dostupné na URL: <http://docs.oracle.com/javaee/6/tutorial/doc/ >. ISSN 821-1841-15.

[9] Armstrong, E. et al. *The J2EE 1.4 Tutorial: For Sun Java System Application Server Platform Edition 8.2*. Oracle, 500 Oracle Parkway , Redwood City, U.S.A. 2005. 5. Decembra 2005. [cit. 10 apríla 2012]. Dostupné na: < http://docs.oracle.com/javaee/1.4/tutorial/doc/JSFIntro10.html >

[10] Heppenstall, D. *Software Metrics*. [cit. 20 apríla 2012]. Dostupné na URL: < http://www.heppenstall.ca/academics/doc/320/L08_Metrics.pdf >.

[11] Genero, M. Piattini, M. Calero, C. *Metrics for software conceptual models*. Imperial College Press, 57 Shelton Street, Covent Garden, Londýn, Veľká Británia, 2005. ISBN 1-86094-497-3.

[12] VanHilst, M. *Developement Process Metrics*. Florida Atlantic University, College of Engineering & Computer science. 22 marca 2007. [cit. 26.4.2012]. Dostupné na URL: < http://www.cse.fau.edu/~security/public/docs/DevelopmentProcessMetrics.ppt >.

[13] Madachy, R. *Software Metrics Guide*. University of Southern California, Los Angeles. [cit. 26.4.2012]. Dostupné na URL: < http://sunset.usc.edu/classes/cs577b_2001/metricsguide/metrics.html#p31>.

Zoznam príloh

Príloha 1: Užívateľská dokumentácia
Príloha 2: CD so zdrojovými súbormi aplikácie a textu

Príloha 1 – Užívateľská dokumentácia

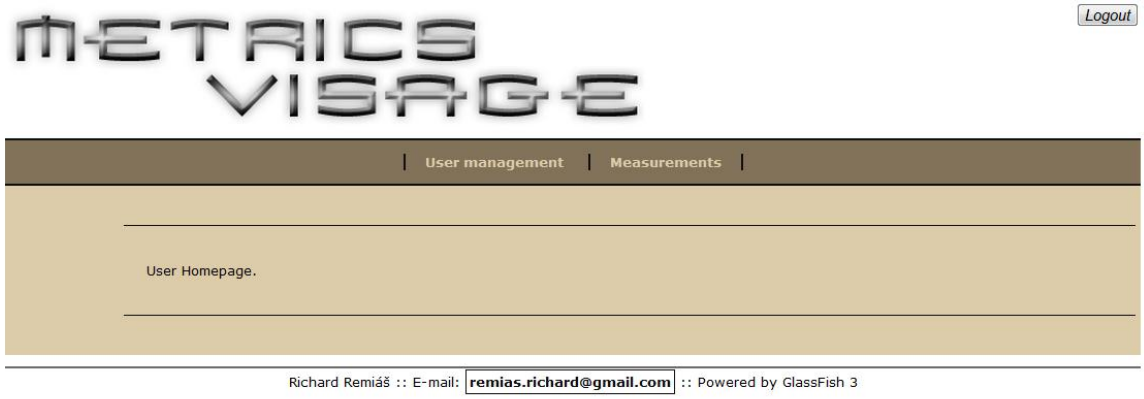
Užívateľská dokumentácie aplikácie MetricsVisage obsahuje popis užívateľského rozhrania aplikácie a postup jej používania. Aplikácia je na základe autentizácie rozdelená na tri úrovne:

- User
- Manager
- Admin

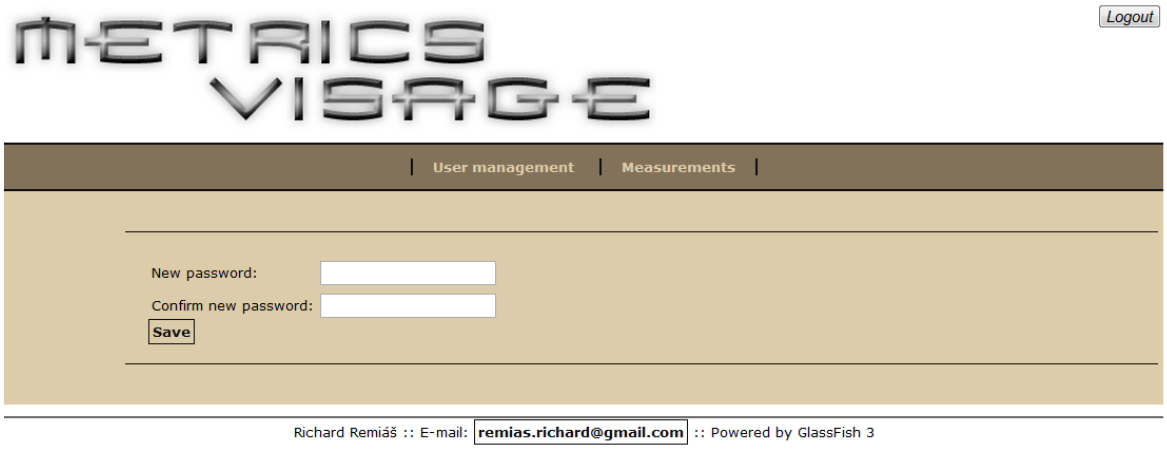
Rozhrania a funkčnosť jednotlivých úrovní je popísaná v nasledujúcich kapitolách.

Skupina user

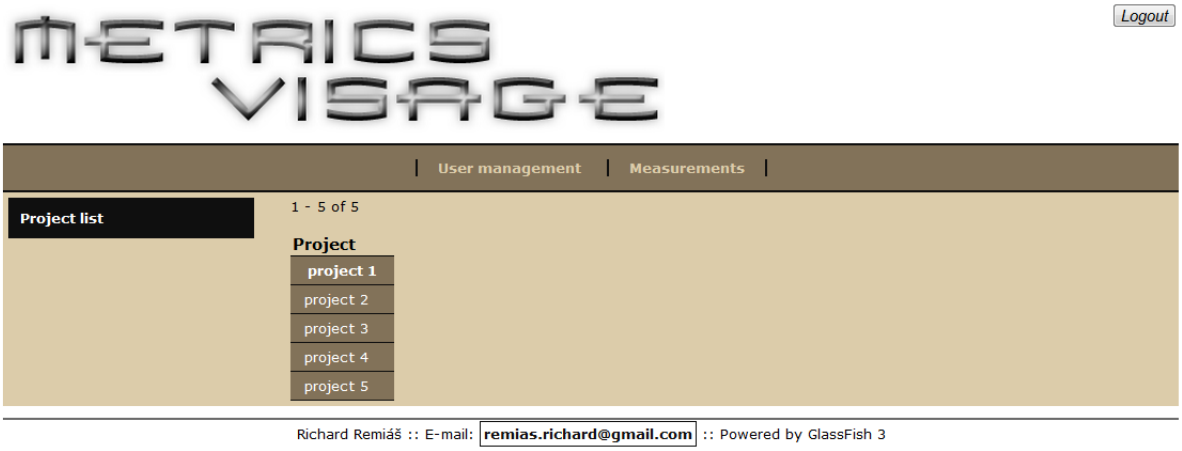
Táto skupina má najmenšie práva. Člen skupiny user má po prihlásení do systému dve možnosti (obr. 2) „User management“ umožňuje užívateľovi zmeniť si svoje prístupové heslo (obr. 3), „Measurements“ zobrazí užívateľovi zoznam všetkých vedených projektov (obr. 4). Užívateľ na úrovni user môže pristupovať len k projektom, na ktorých on sám pracuje.



Obr. 2: Užívateľská úvodná stránka.



Obr. 3: Zmena hesla užívateľa.

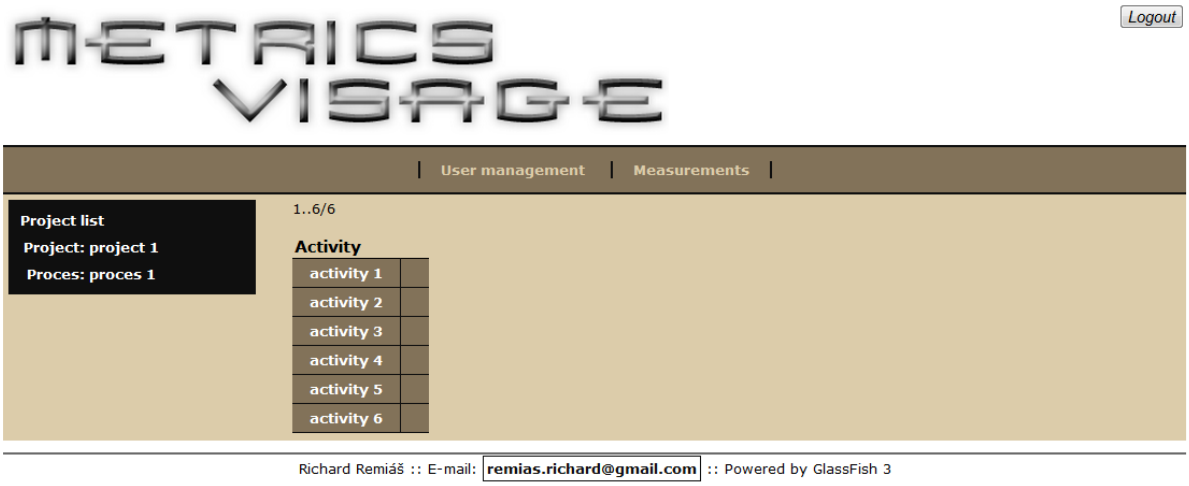


Obr. 4: Zoznam všetkých projektov.

Pri zvolení vybraného projektu sa užívateľovi zobrazí zoznam všetkých procesov v tomto projekte (obr. 5). Vľavo sa nachádza kontextové menu, ktoré užívateľovi umožňuje navigovať sa medzi stránkami, a súčasne mu ukazuje, kde sa momentálne nachádza. Tzn. Na najvyššej úrovni je vo výbere projektov (Project list) a má vybratý projekt s názvom project 1 (Project: project 1). Jednotlivé procesy sú zobrazené spolu s ich vlastníkmi.

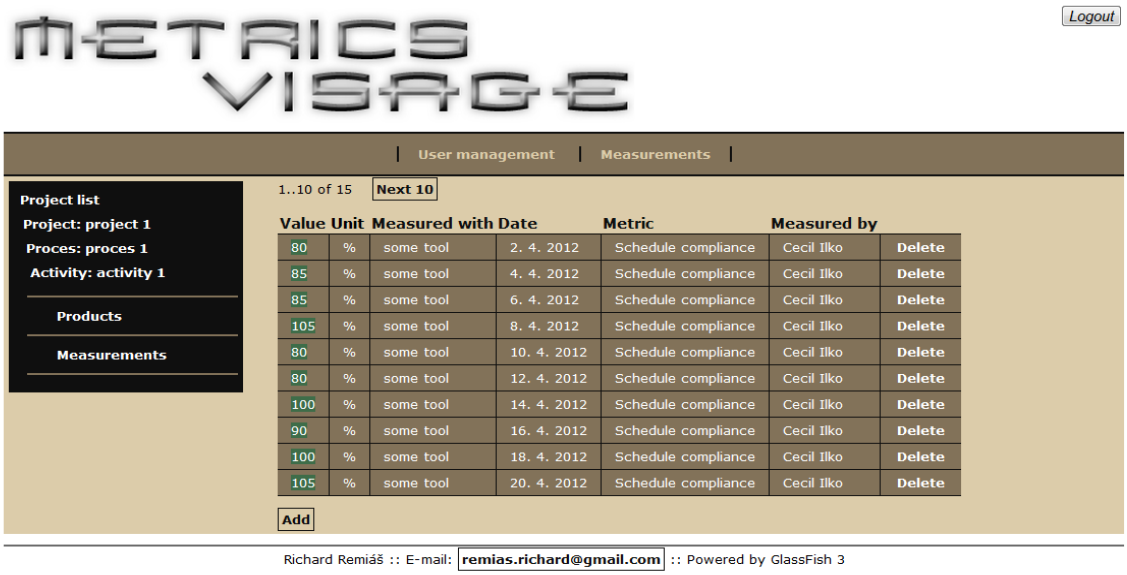
Obr. 5: Zoznam procesov pre vybraný projekt.

Pri zvolení procesu sa užívateľ dostáva o úroveň nižšie na výber aktivít v rámci zvoleného procesu (obr. 6). Pri zvolení aktivity sa užívateľovi zobrazí výber produktov pre danú aktivitu (obr. 7). Okrem toho v kontextovom menu pribudli dve položky: „Products“ je odkaz na zoznam produktov a „Measurements“ je odkaz na zoznam meraní pre vybranú aktivitu (obr. 8). Merania sú zobrazené formou tabuľky. Pre každé meranie sú uvedené: nameraná hodnota, jednotka meranej hodnoty, nástroj použitý na meranie, dátum merania, metrika, ku ktorej sa meranie vzťahuje, užívateľ, ktorý meranie vykoval. Pri meraniach existuje aj možnosť meranie zmazať (napríklad v prípade chyby). Právo na zmazanie merania má iba osoba, ktorá toto meranie vykonala, alebo osoba zo skupiny admin.



Obr. 6: Zoznam aktivít pre vybraný proces.

Obr. 7: Zoznam produktov pre vybranú aktivitu.

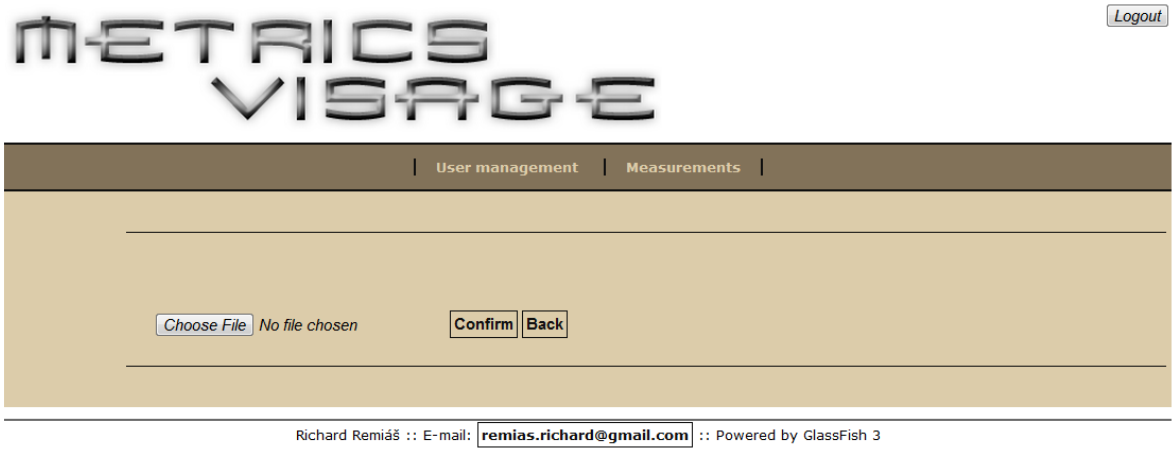


Obr. 8: Zoznam meraní pre vybranú aktivitu.

Pod tabuľkou meraní je odkaz na možnosť pridanie nového merania. Tento odkaz prenesie užívateľa na dialóg pridávania nového záznamu merania (obr. 9). V tomto dialógu užívateľ navolí nameranú

hodnotu (celé číslo), nástroj, pomocou ktorého bolo meranie vykonané, dátum merania a metriku, ku ktorej sa meranie vzťahuje. Meranie sa môže vzťahovať len k metrikám, ktoré sú merané na procese, ktorému aktivita (produkt) patrí. Na úrovni user, je vykonávateľ merania automaticky nastavený na prihláseného užívateľa (nie je možné pridávať merania v mene niekoho iného). Pridávanie meraní obsahuje aj možnosť načítania údajov zo súboru (obr. 10). Aplikácia dokáže načítať a spracovať údaje vo formáte .csv (oddeľovač „;“). Súbor musí obsahovať práve 5 stĺpcov pomenovaných: *date*, *value*, *metric*, *measurer*, *tool* v ľubovoľnom poradí. Formát dátumu musí byť *dd.MM.yyyy*.

Obr. 9: Dialóg pridávania meraní.

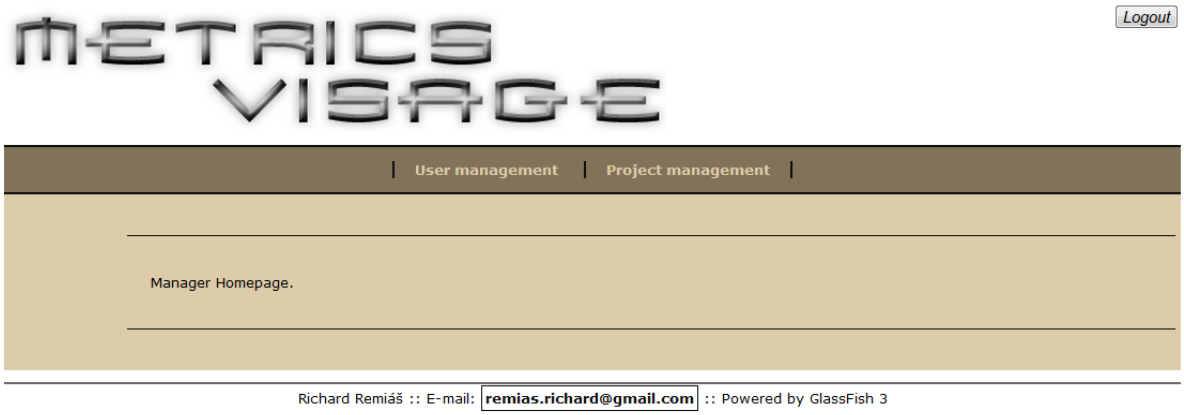


Obr. 10: Dialóg nahrania súboru.

Rozhranie pre pridávanie a zobrazovanie meraní pre produkty aktivít je zhodné s rozhraním pre aktivity.

Skupina manager

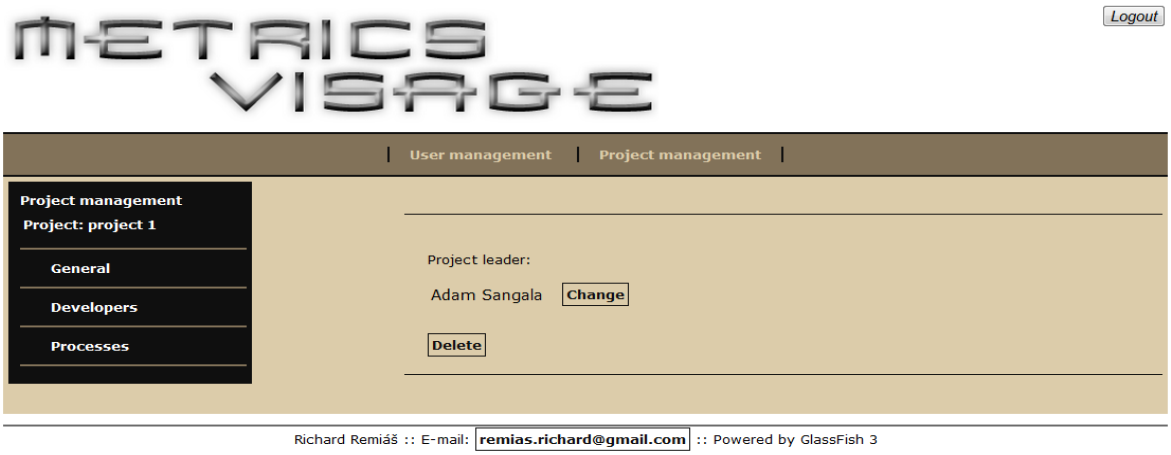
Práva a možnosti skupiny manager sú oproti skupine user rozšírené o možnosť grafického zobrazenia dát z meraní a manažmentu projektov. Úvodná obrazovka pre skupinu manager obsahuje 2 voľby: „User management“, ktorá podobne ako pri skupine user umožňuje užívateľovi zmeniť svoje heslo a „Project management“, ktorá poskytuje správu projektov a meraní (obr. 11, 12). Užívateľ skupiny manager môže naproti tomu zo skupiny user pristupovať ku všetkým projektom a metrickým údajom z týchto projektov. Meniť avšak môže iba projekty, v ktorých má úlohu „leadra“.



Obr. 11: Úvodná obrazovka skupiny manager.

Obr. 12: Zobrazenie projektov.

Po vybratí konkrétneho projektu sa manažérovi zobrazí správcovský kontext pre daný projekt (obr. 13). Na stránke „General“ je uvedený „leader“(manažér) projektu. Editovať manažéra má právo iba manažér projektu, avšak ak zmení manažéra, stratí všetky práva na editovanie projektu. Stránka „Developers“ zachycuje všetkých ľudí momentálne pracujúcich na projekte (obr. 14). Iba užívatelia, ktorí pracujú na danom projekte môžu pridávať merania do projektu, respektíve k nemu pristúpiť. Manažér projektu má možnosť užívateľov z projektu odstrániť, respektíve ich pridať. Ak sa manažér rozhodne pridať ďalšieho užívateľa na projekt, otvorí sa mu dialóg pridávania (obr. 15). Tu sa zobrazí tabuľka všetkých užívateľov, ktorí na projekte doteraz nepracujú, tým sa predíde viacnásobnému pridaniu rovnakého užívateľa do projektu.



Obr. 13: Správa projektu.

Obr. 14: Správa užívateľov pracujúcich na projekte.

Obr. 15: Pridanie užívateľa na projekt.

Na stránke „Processes“ sú zobrazené všetky procesy daného projektu (obr. 16). Jednotlivé procesu sú farebne rozlíšené, podľa „zdravia“ procesu. Zdravie procesu udáva, aké percento meraní na procesov je v prijateľnej zóne. Prijateľná zóna je nadefinovaná pre každú dvojicu procesu a metriky formou spodnej a vrchnej hranice. Ak sa všetky merania pre daný proces a každú metriku na ňom meranú

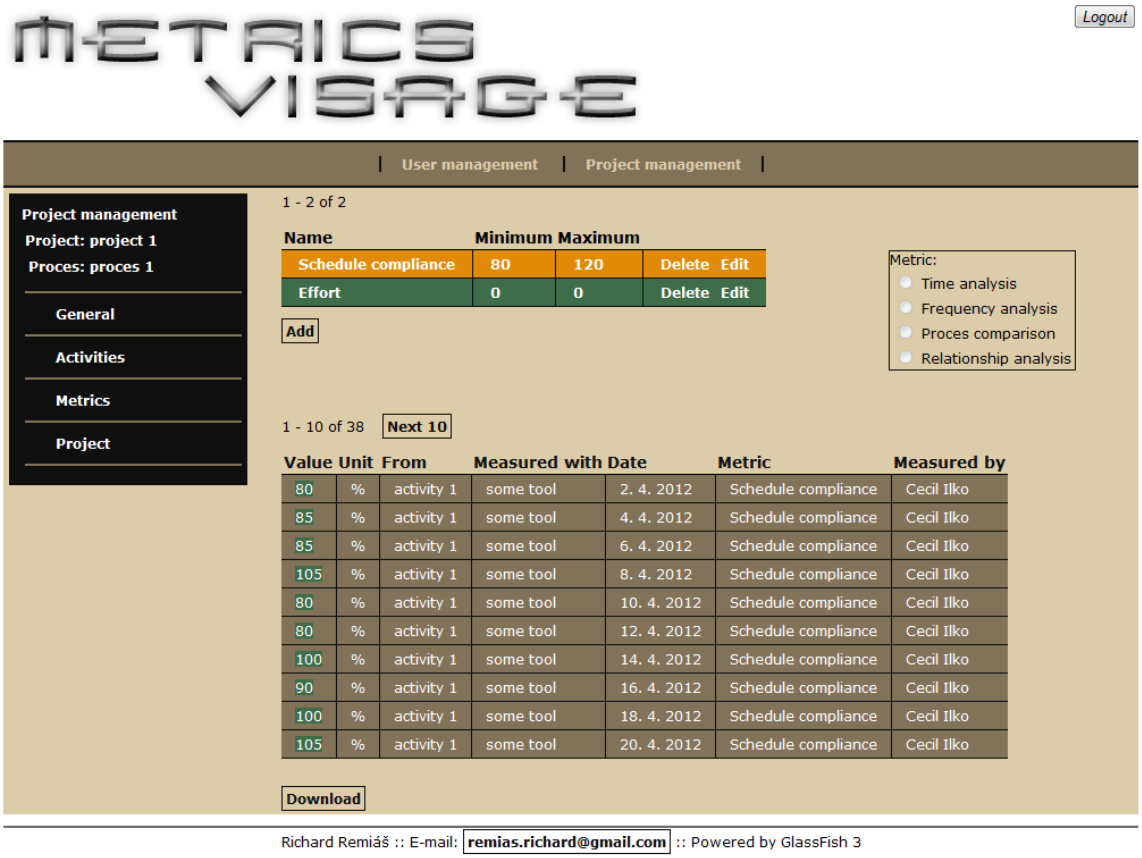
nachádzajú v prijateľnej zóne je proces zelený – zdravý. Ak je počet meraní v prijateľnej zóne nad 50%, tak je proces žltý – nie úplne zdravý. Ak sa viac ako 50% meraní pohybuje mimo prijateľnej zóny, tak je proces červený – chorý a treba mu venovať zvýšenú pozornosť. Ak pre daný proces a metriku nie je definovaná prijateľná zóna (minimum aj maximum je rovné 0) všetky merania týkajúce sa vybranej metriky sú vyhodnotené, ako keby sa nachádzali v prijateľnej zóne. Toto rozdelenie je aplikované aj na jednotlivé aktivity, produkty a metriky v rámci procesu.

Obr. 16: Zoznam procesov vybraného projektu.

Pri vybraní konkrétneho procesu sa zobrazia detaily tohto procesu podobne ako pri projekte (obr. 17). Sú zobrazené údaje o názve procesu, o jeho význame, o jeho vlastníkovi a jeho popis. Ak je prihlásený manažér manažérom daného projektu, môže tieto údaje meniť pomocou tlačidla „Change“. Ak prihlásený užívateľ nie je manažérom daného projektu, toto tlačidlo sa mu nezobrazí.

Obr. 17: Informácie o vybranom procese.

Pre vybraný proces je možné zobraziť aktivity daného procesu (voľba „Activities“), metriky daného procesu (voľba „Metrics“) alebo sa navrátiť späť na projekt (voľba „Project“).Ak sa rozhodne pre vybraný proces zobraziť metriky, zobrazí sa stránka metrík (obr. 18). Ak je prihlásený užívateľ manažérom vybraného projektu, môže metriky pre proces pridávať (obr. 19) alebo odstraňovať, prípadne upraviť prijateľnú zónu. Ak sa rozhodne metriku odstrániť, **stratia sa s ňou všetky merania, na danom procese, ktoré patria odstraňovanej metrike**.



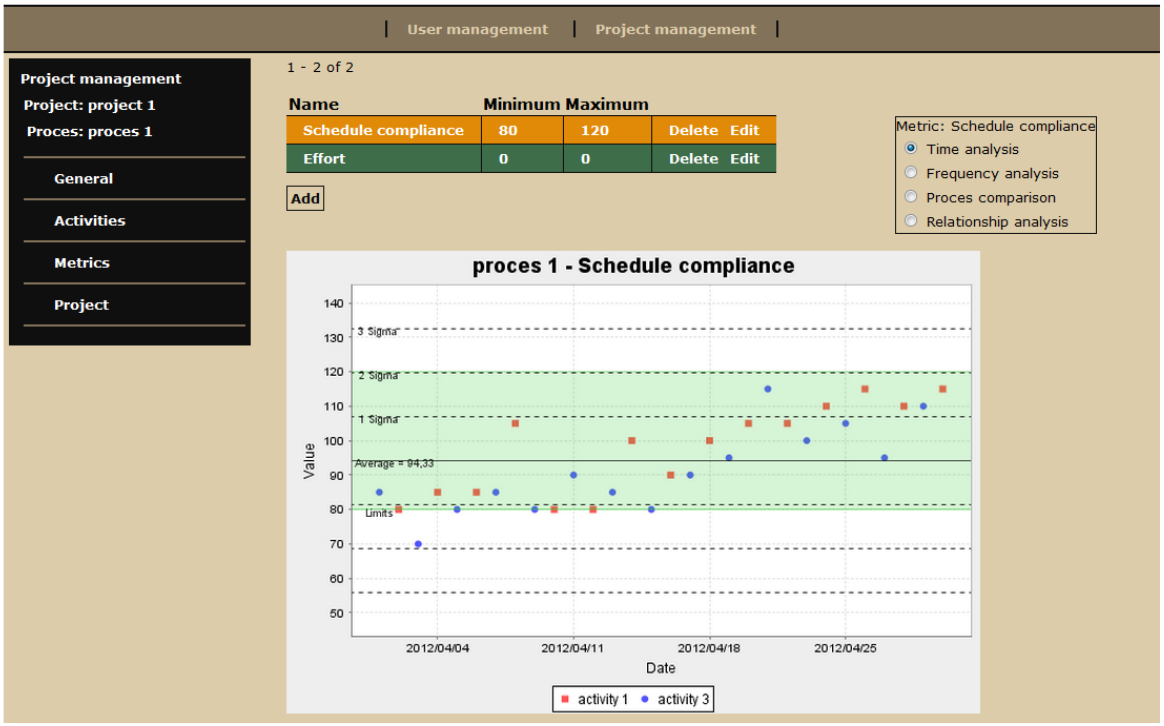
Obr. 18: Zobrazenie metrík pre vybraný projekt.

Obr. 19: Pridanie novej metriky pre proces.

V spodnej časti je zobrazená tabuľka všetkých meraní pre zvolený proces. Ak užívateľ vyberie konkrétnu metriku, tento zoznam bude upravený na merania týkajúce sa vybranej metriky a vybraného procesu. Zároveň bude užívateľovi umožnené grafické zobrazenie metrických dát. V pravom hornom rohu obrazovky je menu pozostávajúce zo štyroch položiek: *Time analysis*, *Frequency analysis*, *Proces comparison* a *Relationship analysis*. Po zvolení metriky sa vo vrchnej časti zobrazí názov zvolenej metriky a toto menu sa stane aktívnym.

Time analysis

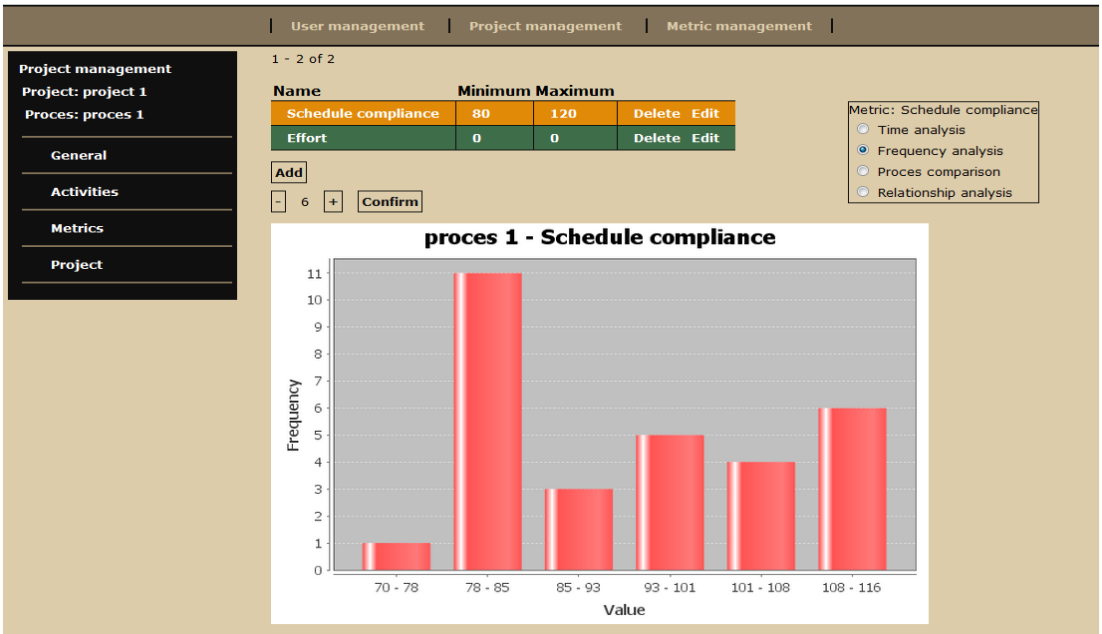
Pri časovej analýze procesu sa zobrazí XY graf. Jeho názov tvorí názov vybraného procesu a metriky. Jeho X-ová osa reprezentuje časovú doménu. Na Y-ovej osi sú vynesené hodnoty meraní. Každý bod predstavuje jedno konkrétne meranie (obr. 20). V našom prípade tvoria body dve série. Každá séria zodpovedá jednej konkrétnej aktivite (produktu aktivity) a je farebne a tvarovo odlíšená od ostatných. V ukážkovom prípade sa jedná o merania z aktivít: activity 1, activity 3 (popísané v legende grafu). Okrem samotných bodov sú v grafe zobrazené aj viaceré hranice. Plnou čiernou čiarou je zobrazený priemer (Average) pre daný súbor meraní, ktorý vyjadruje priemernú hodnotu meraní pre vybraný proces a metriku. Prerušované čiary predstavujú hodnotu jedno, dvoj a trojnásobku štandardnej odchýlky (Sigma). Zelené rozmedzie predstavuje prijateľné rozmedzie určené pre vybranú metriku a proces.



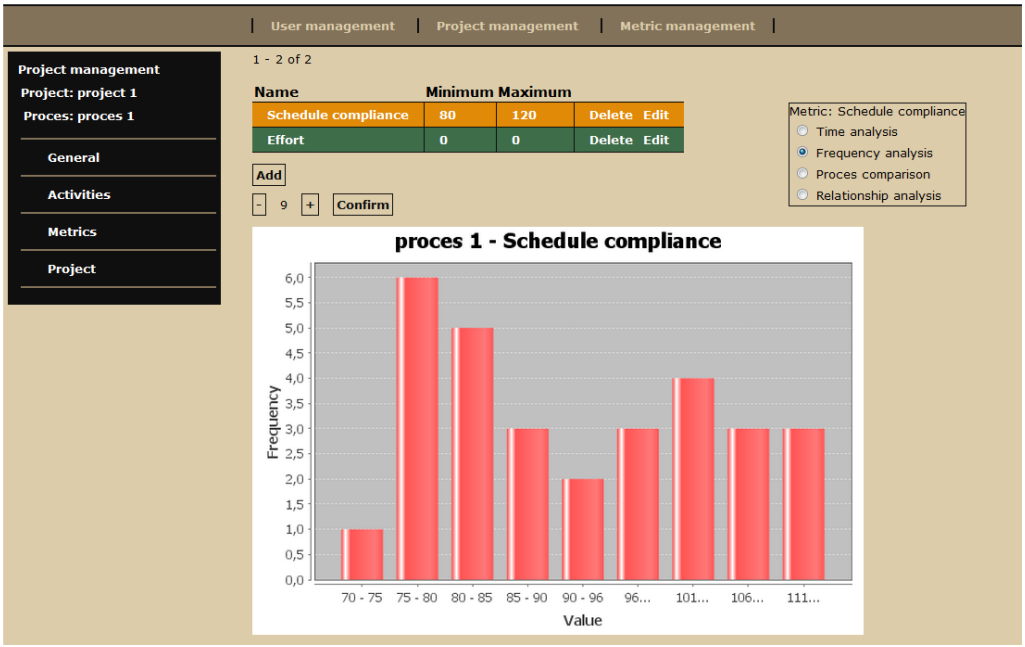
Obr. 20: Časová analýza procesu.

Frequency analysis

Frekvenčná analýza zobrazuje merania v stĺpcovom grafe (obr. 21). Jednotlivé stĺpce predstavujú rozmedzia hodnôt meraní. Výška stĺpca určuje, koľko meraní sa nachádza v rozmedzí hodnôt pre daný stĺpec. V grafe je možné meniť počet stĺpcov – na to slúžia tlačidlá „+“ a „-“. Po navolení požadovaného počtu stĺpcov sa tlačidlom „Confirm“ potvrdí nový počet a prekreslí sa graf. Počiatočná hodnota je nastavená ako druhá odmocnina rozdielu maximálnej a minimálnej hodnoty v grafe.



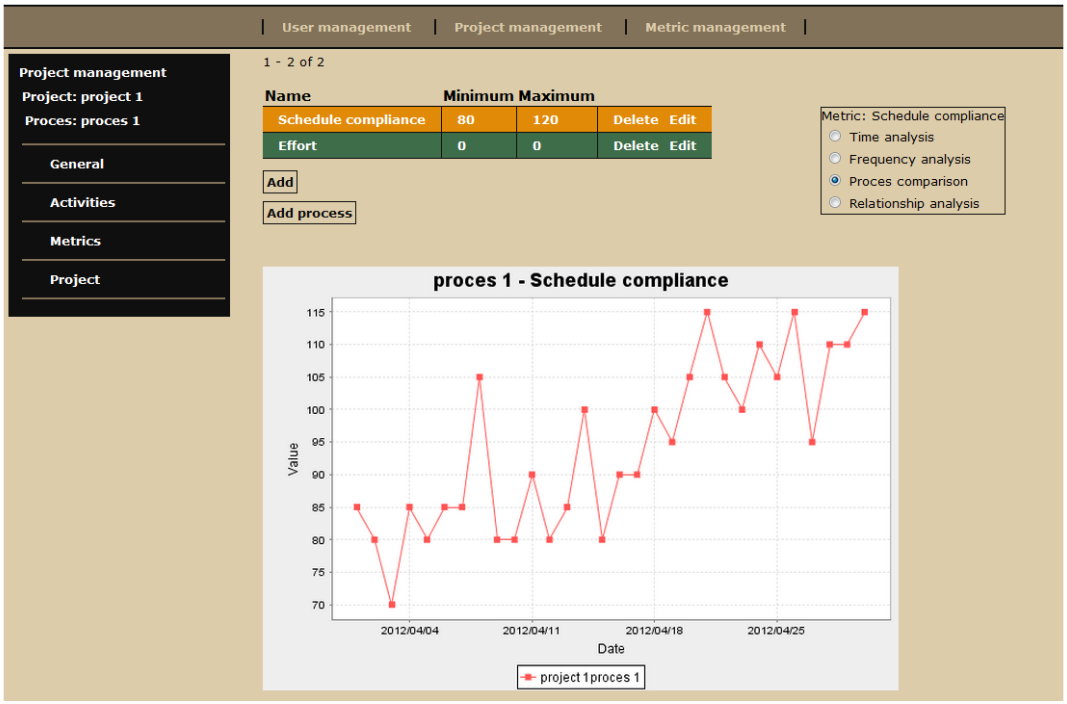
Obr. 21: Histogram so 6 stĺpcami.



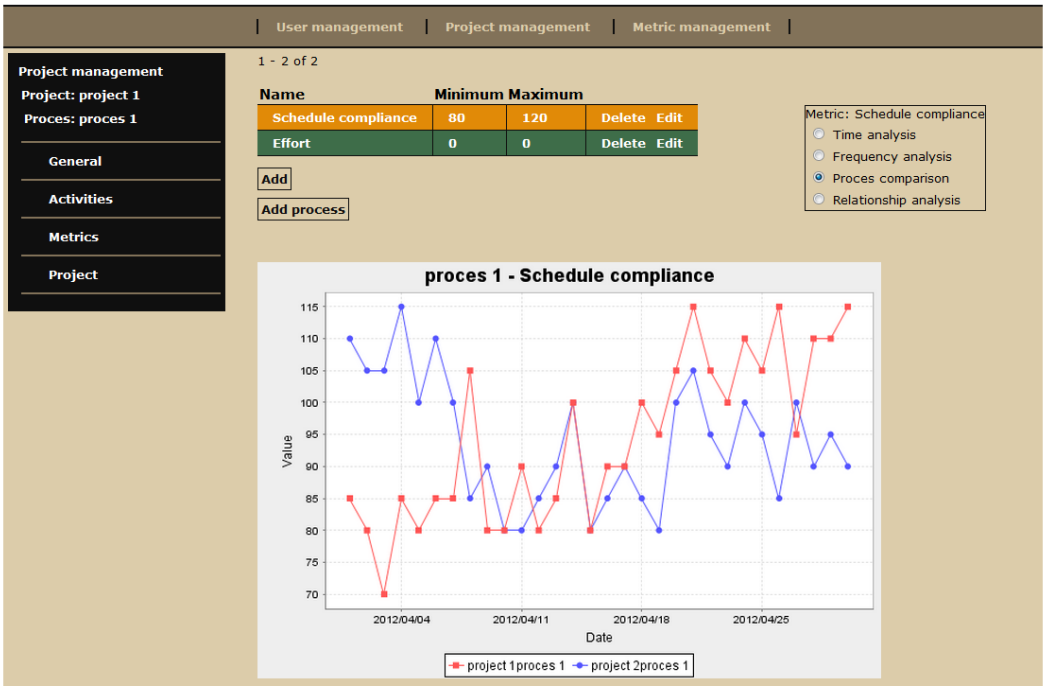
Obr. 22: Histogram s 9 stĺpcami.

Proces comparison

Porovnávanie procesov nám umožňuje porovnať dva alebo viac odlišných procesov z rozličných projektov. Pri základnom zobrazení je zobrazený časový priebeh vybraného procesu pre vybranú metriku (obr. 23).



Obr. 23: Porovnávanie procesov – vykreslený jeden proces.

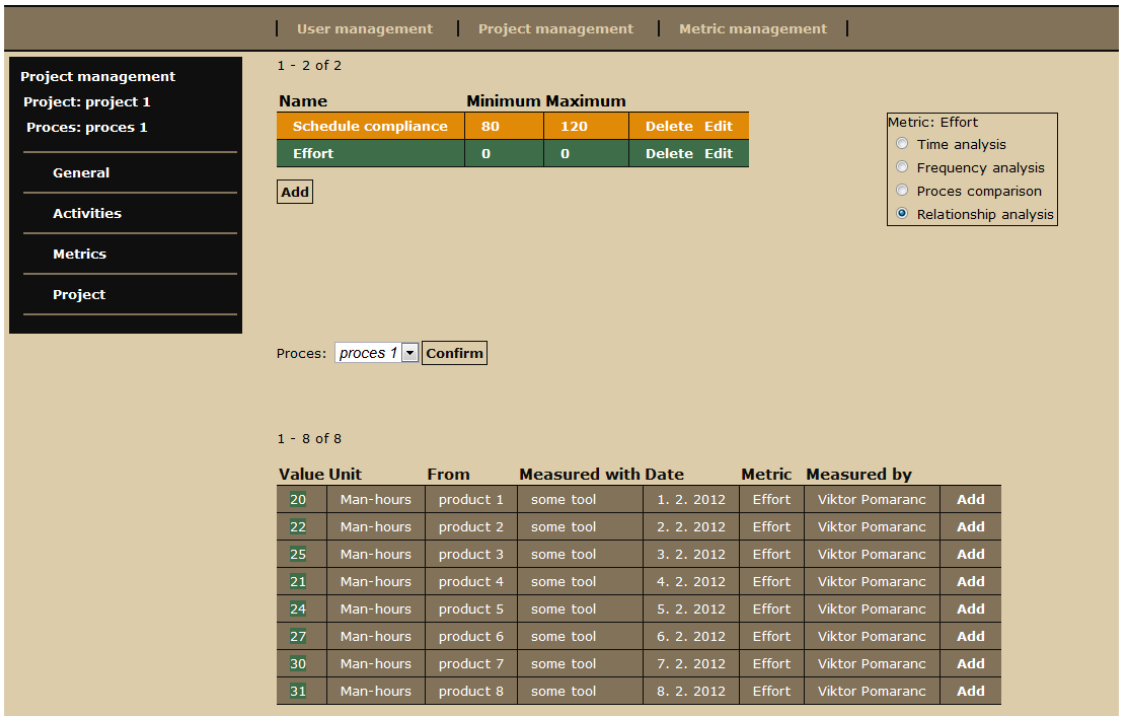


Obr. 24: Porovnávanie procesov, vykreslené dva procesy.

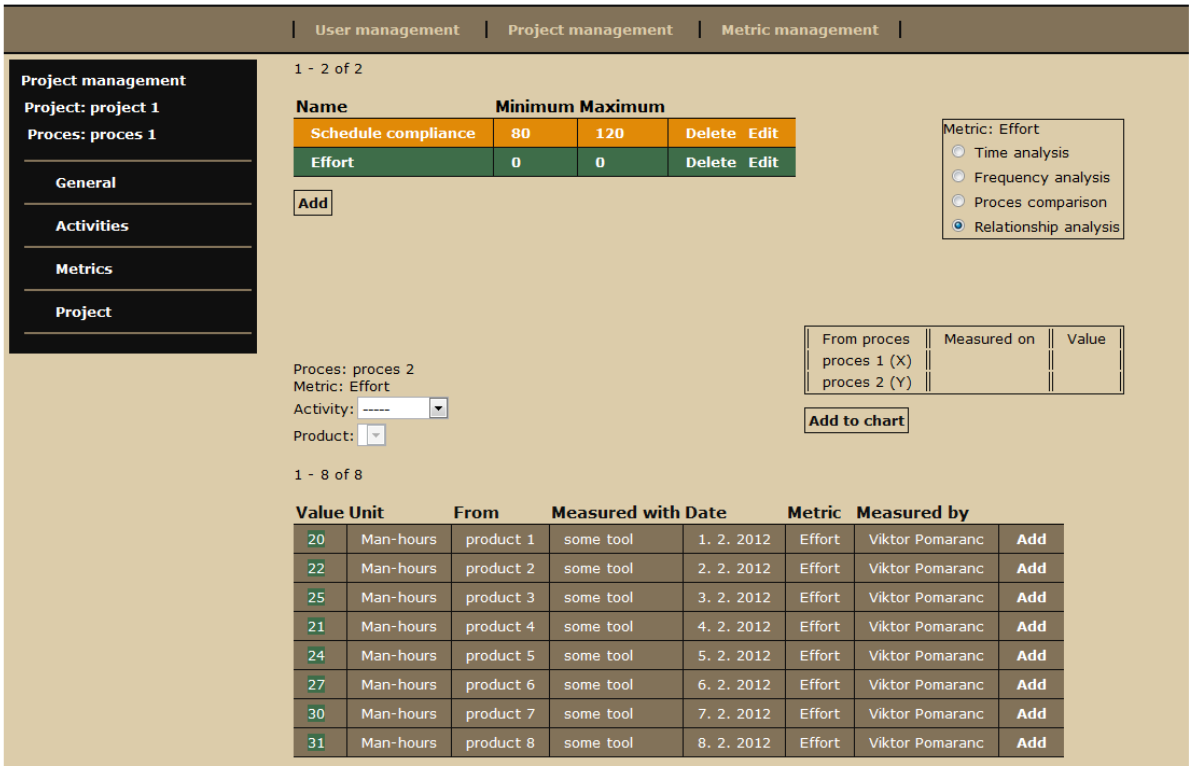
Voľbou „Add process“ je možné do grafu pridať údaje z meraní pre ďalší zvolený proces. Do grafu je možné pridávať iba tie procesy, na ktorých je meraná rovnaká metrika ako konkrétne zvolená metrika pre zvolený proces (obr. 24). Pridávať je možné aj procesy z iných projektov. Časové osi procesov sa zosúladia podľa prvého zvoleného procesu, aby bolo porovnanie možné.

Relationship analysis

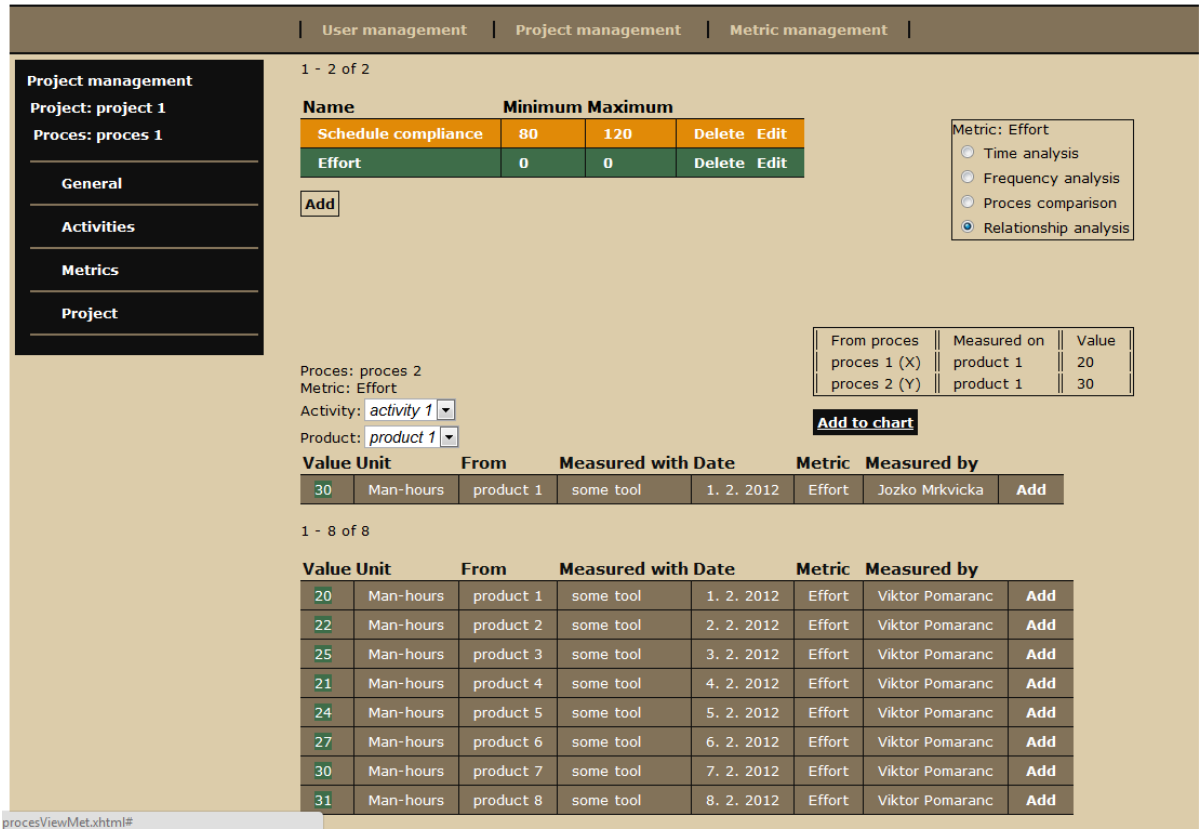
Analýza vzťahov umožňuje užívateľovi zistiť, či medzi dvoma sériami dát existuje nejaká závislosť (obr. 25). Ako jedna séria je vybraná dátová séria meraní pre zvolený proces a zvolenú metriku (zoznam meraní na spodnej časti obrazovky). Druhú dátovú sadu si môže užívateľ navoliť podľa vlastného uváženia. Vzťahy medzi dátami aplikácia umožňuje určovať len v rámci jedného projektu – nie je teda možné merať závislosť medzi procesmi z dvoch odlišných projektov. Užívateľ si najprv vyberie proces zo zvoleného projektu, ktorého vzťah s vybraným procesom chce zistiť (selectbox obsahuje všetky procesy projektu) a svoj výber potvrdí voľbou „Confirm“. Následne pre zvolený proces vyberie metriku, ktorá sa má dávať do vzťahu so vybranou metrikou na vybranom procese. Po zvolení metriky a procesu je situácia ako na obrázku 26. Pre zvolený proces a metriku si môže užívateľ vyberať jednotlivé aktivity a ich produkty, ktorých meranie chce zobraziť. Ďalej pribudol nový prvok, ktorý slúži na pridávanie samotných bodov do grafu. V tejto dvoj-riadkovej tabuľke je uložená x-ová a y-ová súradnica bodu grafu. Jednotlivé súradnice je možné pridať voľbou „Add“ v tabuľke zobrazených meraní pre vybrané procesy (obr. 27). Pri zvolení voľby „Add to chart“ je bod pridaný do grafu, graf je zobrazený a zvolené hodnoty sú vynulované. Takto môže užívateľ bod po bode vytvoriť graf pre vzťahovú analýzu. Aplikácia poskytuje užívateľovi pomerne robustné rozhranie pre tvorbu tohto typu grafu, užívateľ teda môže vypracovať vzťahovú analýzu pre ľubovoľné, možno aj nekonzistentné sady dát. Preto by užívateľ, ktorý tvorí takúto analýzu mal mať dostatočné znalosti, aby výsledky analýzy dávali zmysel.



Obr. 25: Základná obrazovka vzťahovej analýzy.



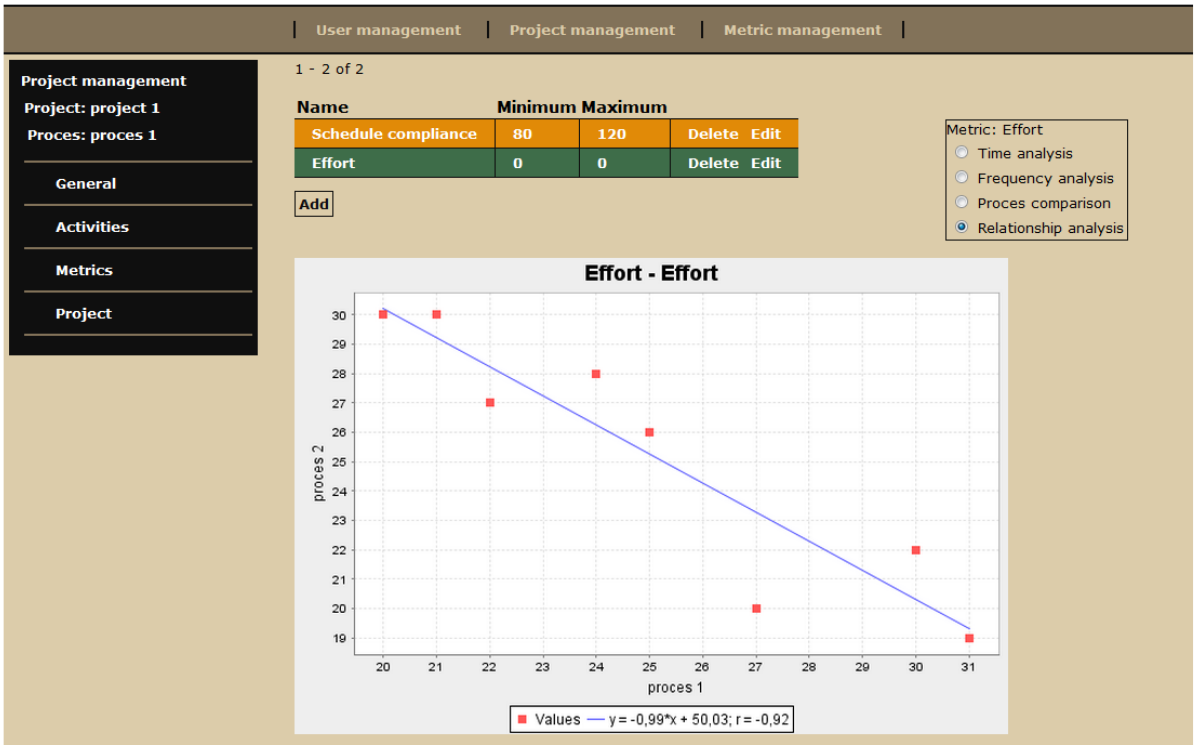
Obr. 26: Vzťahová analýza pri vybranom procese a metrike.



Obr. 27: Pridanie konkrétneho bodu do grafu vzťahovej analýzy.

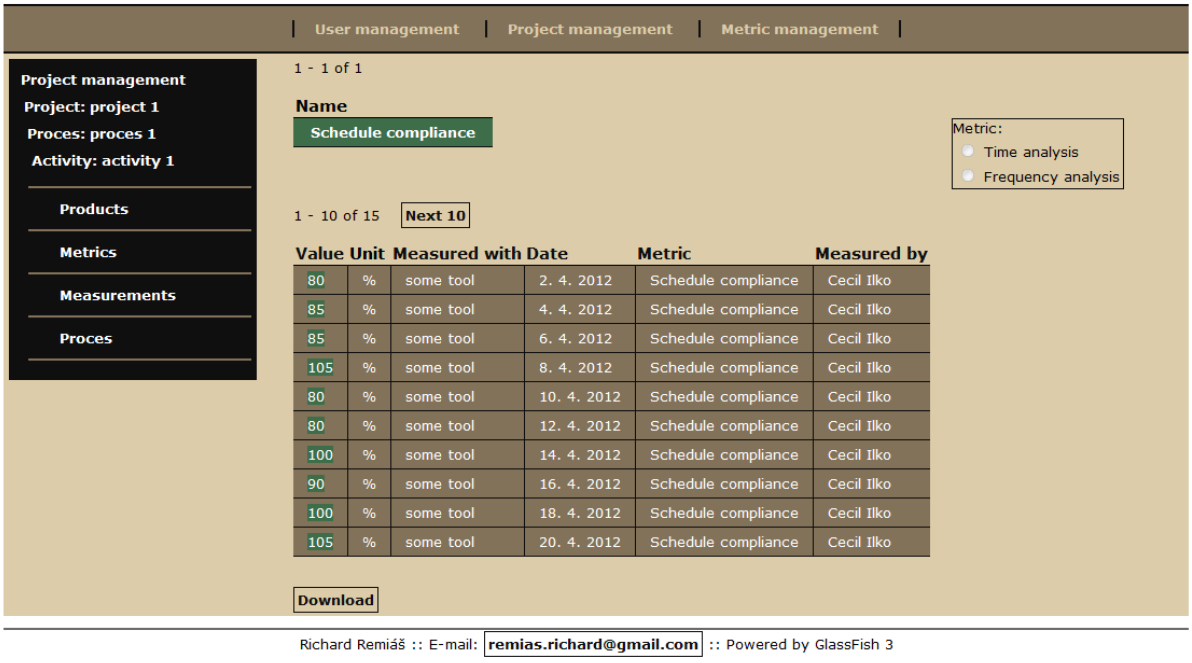
Výsledkom analýzy je graf, na ktorého x-ovej ose ako nezávislá premenná sú hodnoty meraní vybraného procesu na analýzu (v našom prípade *proces 1*) a na y-ovej ose sú hodnoty meraní zo zvoleného procesu vo vzťahovom kontexte (*proces 2*). Jednotlivé body grafu sú teda dvojice

vytvorené užívateľom na základe jeho znalostí a kontextu projektu (obr. 28). Okrem jednotlivých bodov je v grafe vykreslená aj priamka lineárnej regresie, ktorá znázorňuje závislosť vybraných údajov. Rovnica priamky je uvedená v legende grafu ako aj koeficient *r*. Tento koeficient určuje silu závislosti dátových sérií.



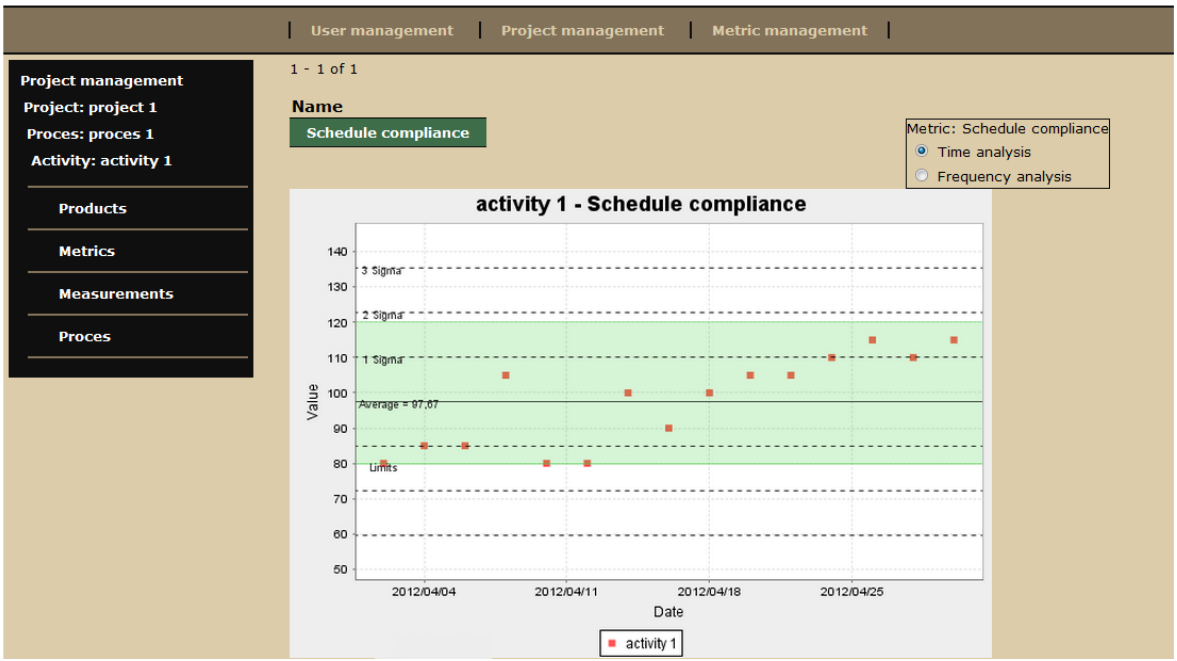
Obr. 28: Kompletná vzťahová analýza.

Rozhranie pre aktivity a produkty procesu je takmer zhodné s rozhraním pre užívateľskú skupinu user. Novinkou je analýza údajov na nižšej úrovni (obr. 29). Oproti skupine user pribudla možnosť analyzovať metrické údaje aj na úrovni aktivít, respektíve ich produktov.

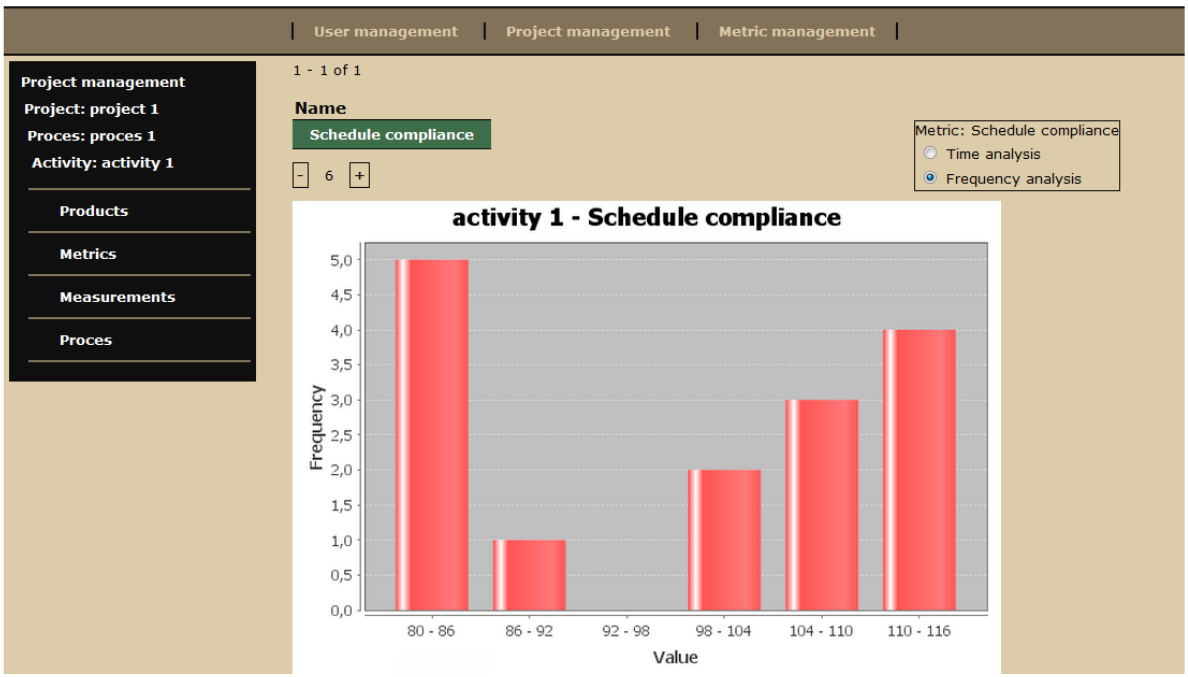


Obr. 29: Analýza údajov na úrovni aktivity.

Tu sú metriky na základe meraní, ktoré boli vykonané na danej aktivite (produkte). Možnosti analýzy údajov sú obmedzené na časovú analýzu (obr. 30) a frekvenčnú analýzu (obr. 31), ktoré sú popísané vyššie v kapitole.



Obr. 30: Časová analýza na úrovni aktivity.

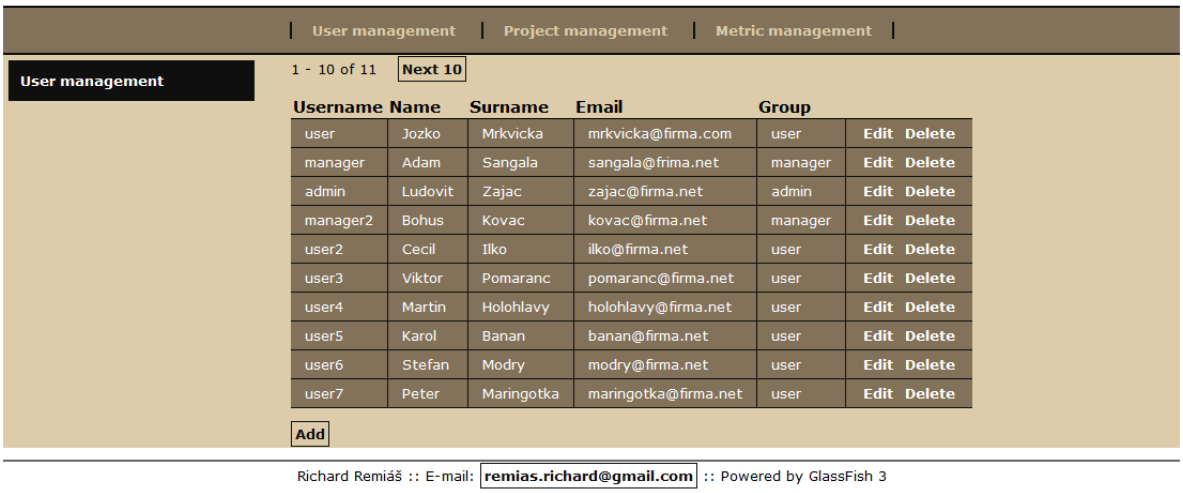


Obr. 31: Frekvenčná analýza na úrovni aktivity.

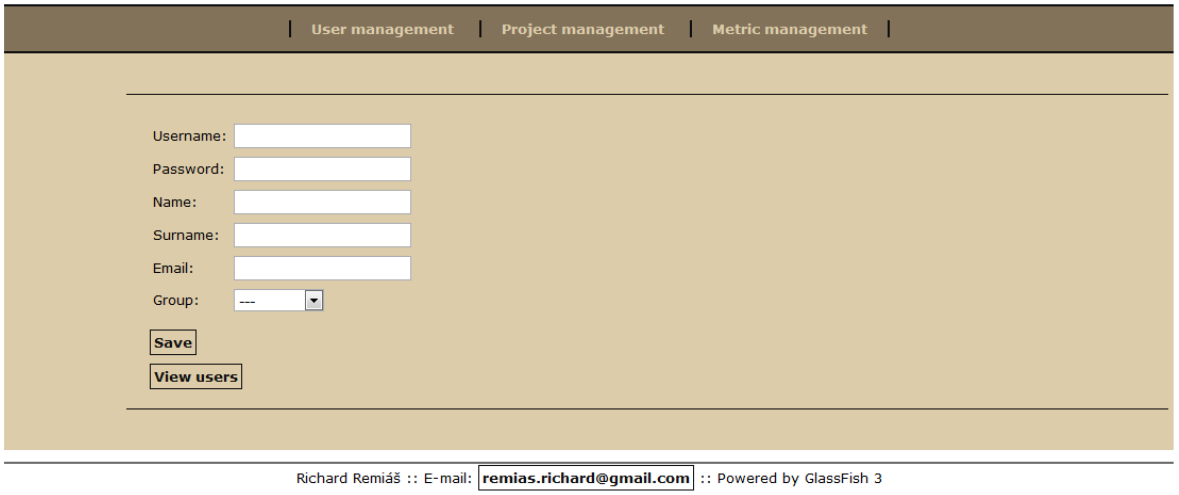
Skupina admin

Skupina admin má najvyššie práva spomedzi všetkých skupín užívateľov. Administrátor má prístup k údajom z všetkých projektov a takisto má právo meniť údaje o všetkých projektoch. Okrem toho má možnosť správy užívateľov a metrík.

Správa užívateľov je vo forme tabuľky (obr. 32). V tabuľke sú zobrazené všetky informácie o užívateľovi (užívateľské meno, meno, priezvisko, email, skupina), okrem jeho hesla. Pridávať užívateľov je možné pomocou voľby „Add“. Pre pridanie užívateľa je otvorený formulár, kde je treba vyplniť potrebné údaje (obr. 33). Pre editáciu užívateľov je použitý podobný formulár s jedným rozdielom: zadávanie hesla. Pri pridaní nového užívateľa je nutné heslo zadať, avšak ak editujeme užívateľa a nezadáme nové heslo je automaticky použité jeho staré heslo. Administrátor má aj možnosť odstrániť užívateľa. Toto je možné iba ak užívateľ nemá vykonané žiadne merania, resp. nepracuje na žiadnom projekte.



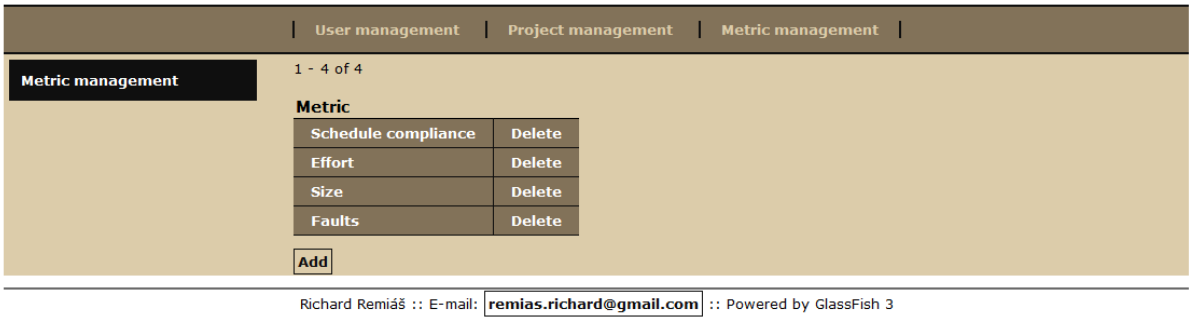
Obr. 32: Obrazovka správy užívateľov.



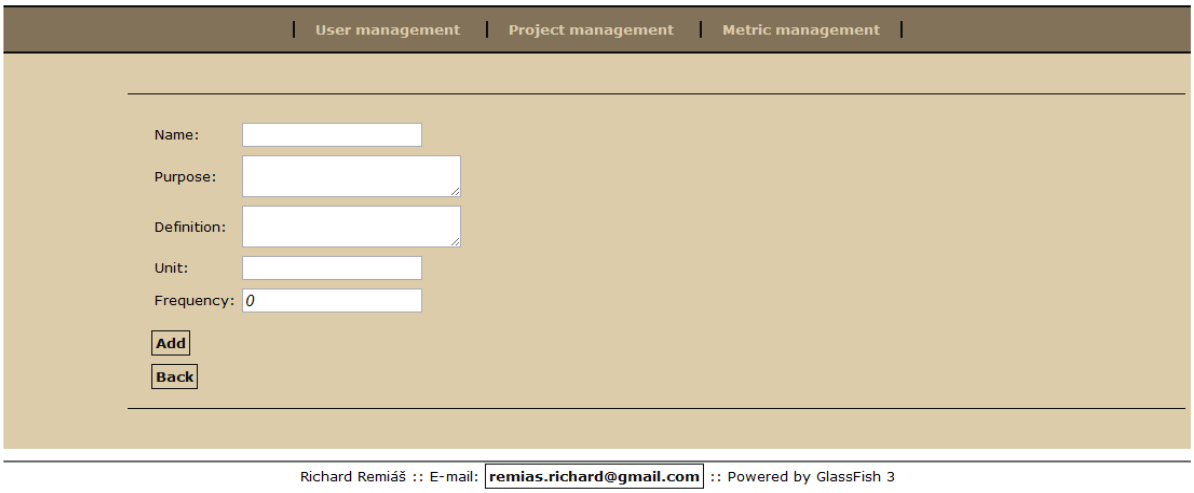
Obr. 33: Dialóg pridania užívateľa.

Správa metrík je obdobná správe užívateľov (obr. 34). Administrátor má možnosť pridávať nové metriky (obr. 35), editovať a odoberať existujúce. Odobrať je možné iba metriky, na ktoré sa neviaže

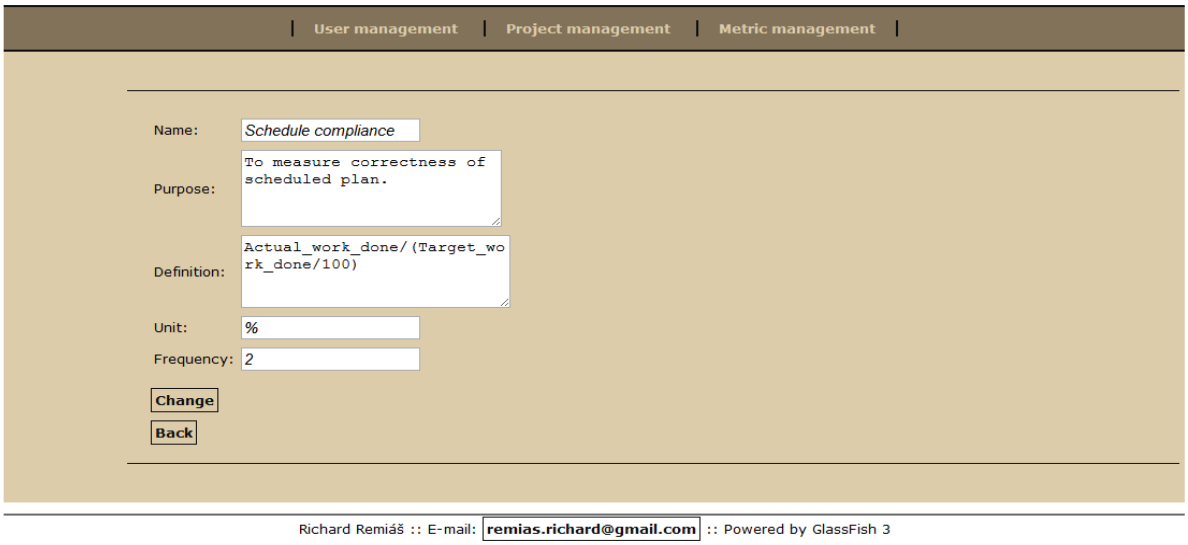
žiadne meranie. Po zvolení konkrétnej metriky sa zobrazia detaily metriky (obr. 36), ktoré je možné meniť voľbou „Change“.



Obr. 34: Obrazovka správy metrík.



Obr. 35: Dialóg pridávania metrík.



Obr. 36: Zobrazenie detailu metriky.